Geospatial and Imagery Access Services Specification

National Imagery and Mapping Agency
United States Imagery and Geospatial System

Release Date: 22 July, 1997
Version 3.0

Acknowledgments

Many individuals and organizations provided support and technical contributions to this work, Individuals from numerous government agencies, contractor organizations and vendors contributed significantly to the development of this specification. We acknowledge these contributions and hope that these individuals and organizations will continue to actively support future updates and extensions. Thanks in advance.

Revision History

- Image Access Facility, Version 0.1 Straw 23 May 1995.
- Image Access Facility, Version 0.2 Tin 11 June 1995.
- Image Access Facility, Version 0.3 Aluminum 19 June 1995.
- Image Access Facility, Version 0.4 Copper - For USIS release June 21, 1995.
- Image Access Facility, Version 0.5 Nickel - Preliminary draft release for Image Access Working Group (IAWG) June 29, 1995.
- Image Access Facility, Version 0.6 Iron - This release contained a relatively complete description of semantics and sequencing for sample implementation prototypers. July 12, 1995.
- Image Access Facility, Version 0.7 Silver - This release addressed comments received. September 6, 1995.
- Image Access Facility and Catalog Access Facility, Version 0.8 Gold - This release contained extensions based upon the additional architecture mining. February 8, 1996.
- Image Access Facility and Catalog Access Facility, Version 0.85 Gold Interim - Update for release and comment on March 22, 1996.
- Image Access Services Specification, Version 0.9 Platinum - Revisions based upon comments from Core Team Working Group, April 24, 1996.
- Image Access Services Specification Version 1.0 - ICCB Configuration-controlled, pilot operational specification for contractor and commercial prototyping and interoperability testing, June 20, 1996.
- Image Access Services Specification Version 1.1 -
  Revised to remove TBR's and TBDs concerning the PNF and IDF. Released for comments December 6, 1996
- Image Access Services Specification Version 1.1 -
  Approved by ICCB December 20, 1996.

- Name of document changed to Geospatial and Imagery Access Services Specification. Version number set to 3.0 to reflect extensions and updates for inclusion of geospatial data and operations.

- Geospatial and Imagery Access Services Specification
  Version 3.0 - Released for NCCB submittal 22 July 1997

Planned Releases

- Geospatial and Imagery Access Services Specification Version 3.1 - Update to incorporate responses and comments from additional interface prototyping tests - Nov. 1997 (TBR)

- Regular updates at approximately six month intervals.

Preface

This document defines common interfaces for the United States Imagery and Geospatial System (USIGS) geospatial and imagery access services.

This specification was prepared consistent with industry practices and is modeled after those being prepared by the Object Management Group (OMG) industry consortium. This approach is consistent with guidelines and direction established by the NIMA Common Imagery Interoperability Working Group (CIIWG).

Table Of Contents

1. Overview

1.1 Background

The Geospatial and Imagery Access Services (GIAS) specification defines the core interfaces of the United States Imagery and Geospatial System (USIGS) libraries for client access to geospatial information. Access is defined to include search, discovery, browsing and retrieval of information and its associated meta-data. Geospatial information is defined to include imagery and imagery-based information, maps, charts and any other data that has a well defined association with a point or area on the Earth.

1.2 Overview

The GIAS interfaces are specified using the OMG Interface Definition Language (IDL). IDL is a language-independent notation for specifying software interfaces. IDL can be readily compiled into software interfaces for various programming languages including C, C++, Ada95, and Smalltalk.

To help the reader assimilate the GIAS interface specification, a series of figures are presented providing varying levels of details concerning the interface's structure and usage. At the highest level of abstraction, the GIAS interface are partitioned into four activity categories: library; request managers; request objects; and callback/product objects. Figure 1 shows how the GIAS interface is structured and what are its activity categories.

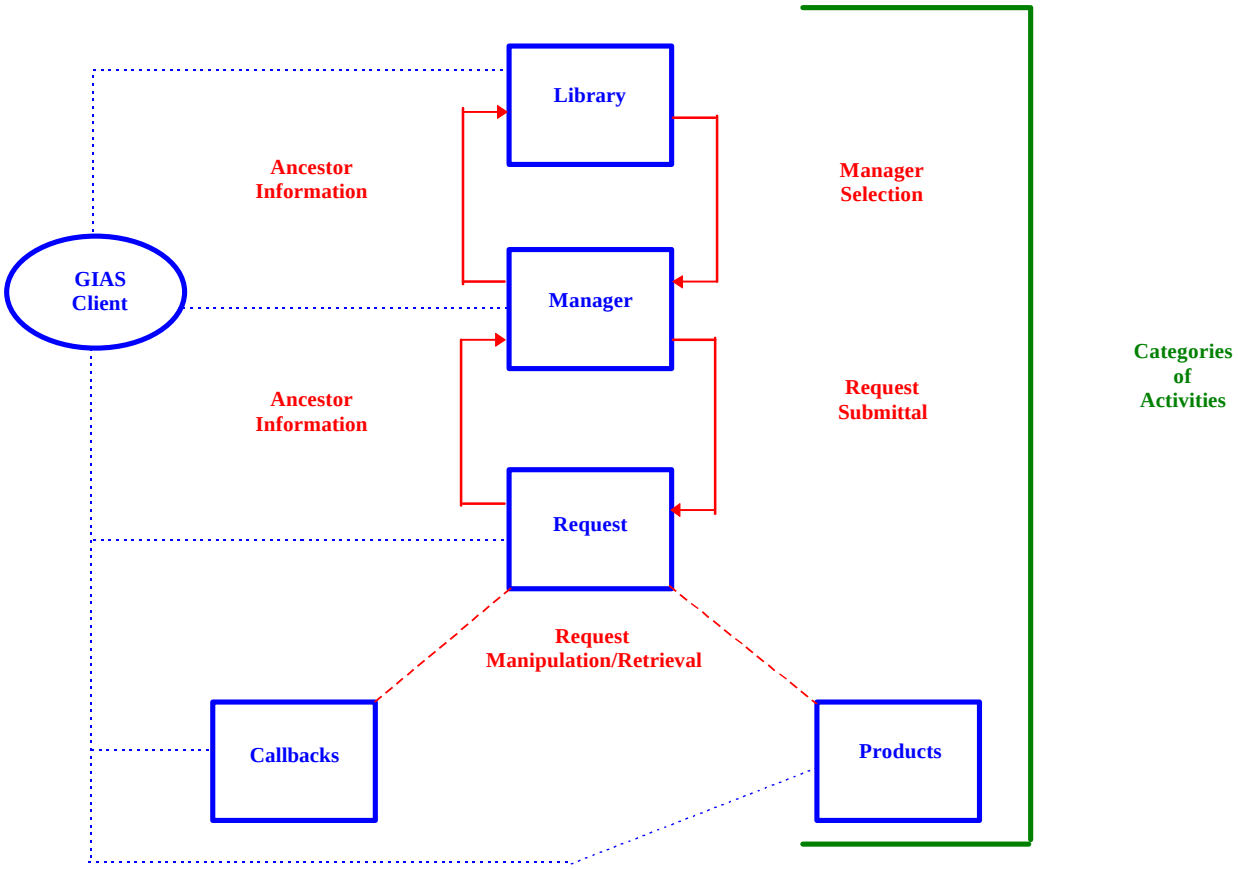

Figure 1 - GIAS Interface Structural and Activity Models

This figure is based on the scenario that a GIAS client requires access to a library, which is accessible through the GIAS interface. The GIAS client interacts with the Library to select and request access to a manager of a specific type. (“manager selection activity category”). Using the provided Manager the client can submit requests for the Library to perform tasks (“request submittal activity category”). Each request submittal returns a Request object. The GIAS client then uses the Request object to monitor progress on the task and to retrieve the results. The Request object also provides a mechanism (a Callback) to allow a client to be notified of the progress of the task. The GIAS client can also obtain information (“ ancestor information activity category”) on a specific request or manager. This allows a GIAS client to determine for any Request, the Manager that is managing it and for any Manager determine the Library(s) it services.

Figure 2 provides another view of the GIAS interface specification as a UML class diagram. As in figure 1, the class diagram is partitioned in four categories of classes: Library; Request Managers; Request Objects; and

Callback/Product Objects. This figure shows the specific operations available on the major objects of each category. It also includes a SampleMgr object and a SampleRequest object to indicate how specific capabilities are produced from the general purpose objects. The GIAS specification includes a number of specific managers and requests based on this model.

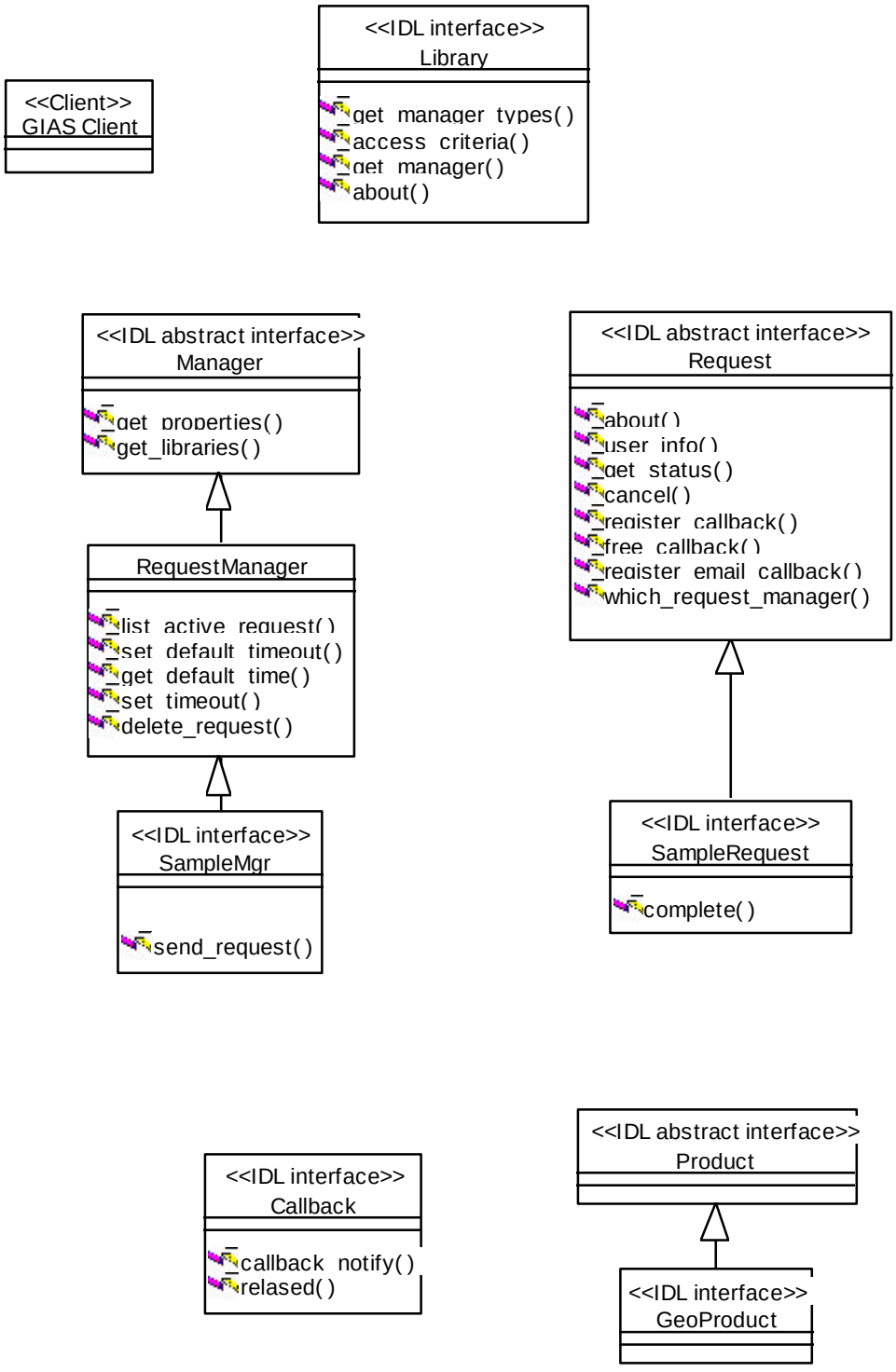

Figure 2 - Generic GIAS Interface Class Diagram

Figure 3 provides a sequence diagram for a generic set of activities (i.e., actions) that would be common for most requests made by GIAS clients. The scenario is initiated by the GIAS Client inquiring and obtaining a list of what types of managers are available from the GIAS Library. Upon receiving and evaluating the list, the GIAS Client selects a manager type (SampleMgr) and request access to a manager object of that type from the GIAS Library. The GIAS Client uses this manager to submit requests (send_request). The SampleMgr returns a SampleRequest object to the client. In this scenario, the GIAS Client is associated with a Callback object. It registers this Callback object with the SampleRequest. When the SampleRequest completes, SampleRequest invokes callback_notify on the Callback object.

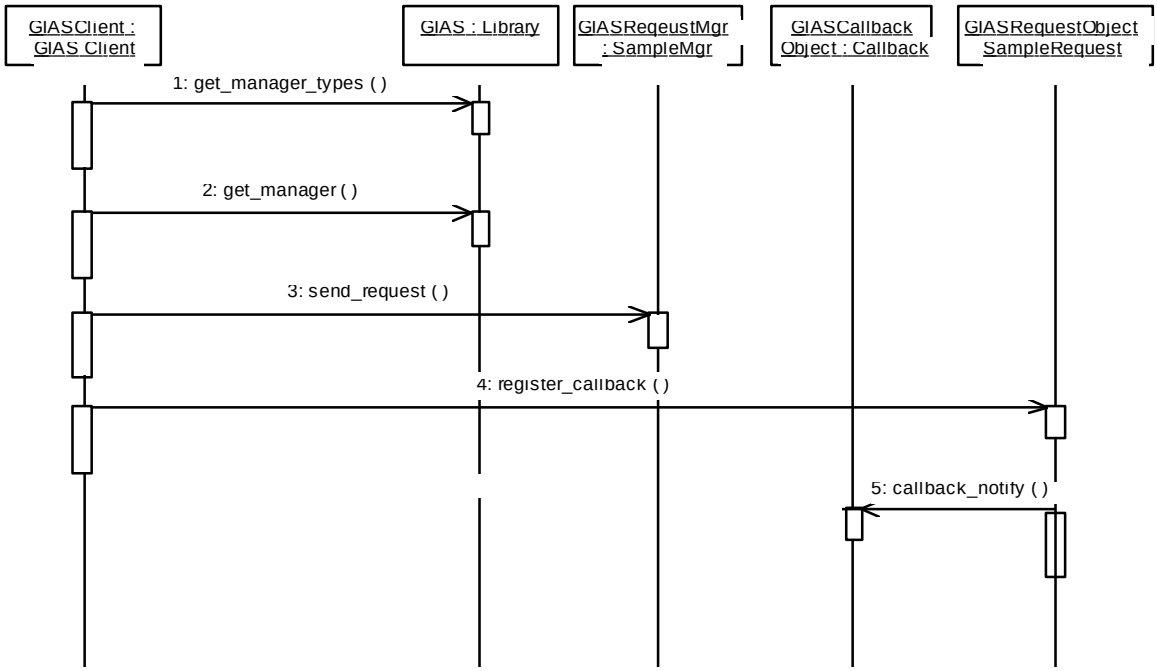

Figure 3 - GIAS Client Generic Request Sequence Diagram

The complete UML class diagram for the GIAS Interface specification described in this document is shown in figures 4 -7 below.

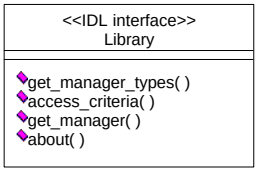

Figure 4 - GIAS Interface Class Diagram (The Library)

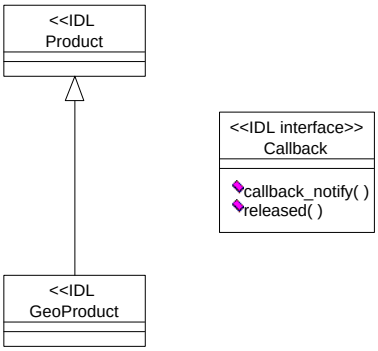

Figure 5 - GIAS Interface Class Diagram (Callback and Product)

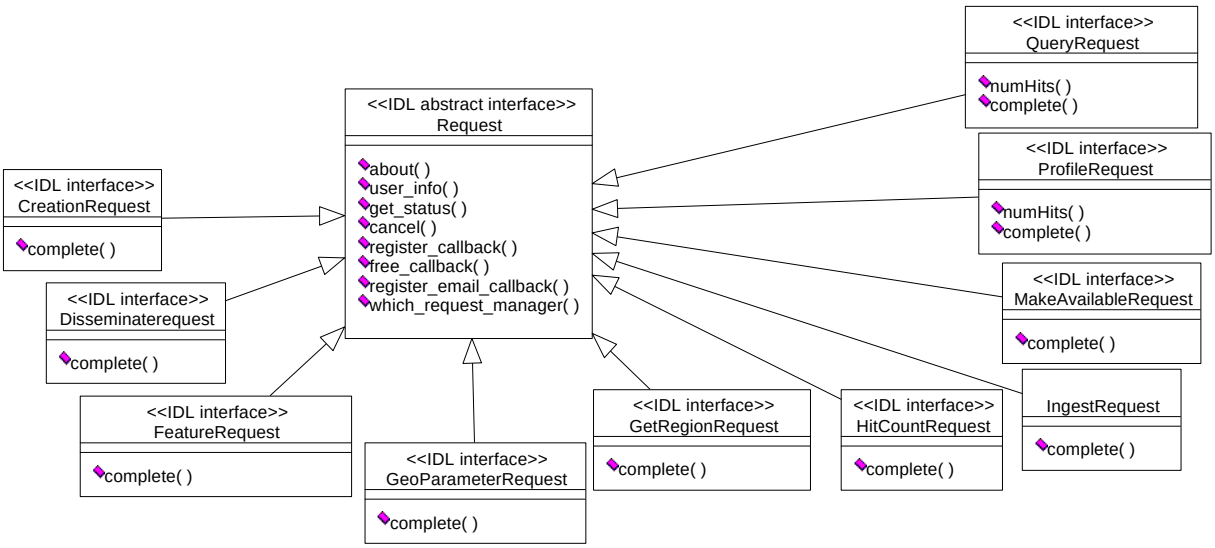

Figure 6 - GIAS Interface Class Diagram (The Requests)

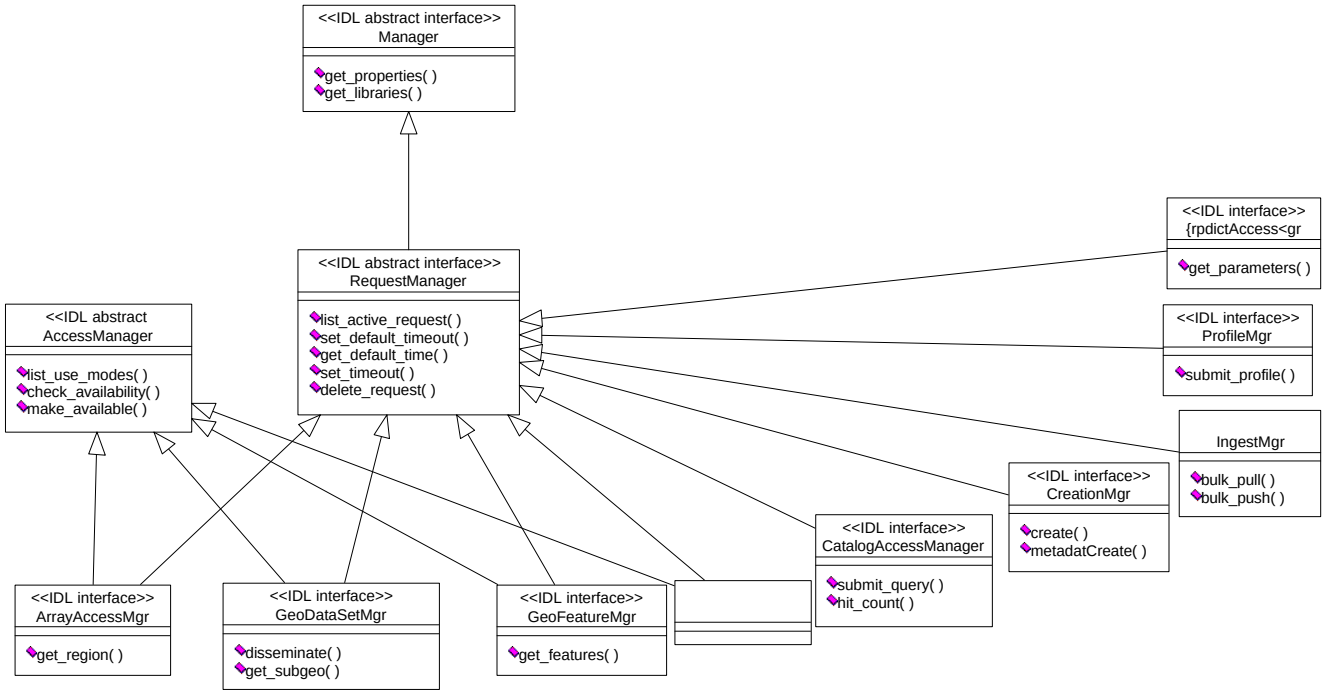

Figure 7 - GIAS Interface Class Diagram
(The Managers)

2. Interface Overview

2.1 Overview

The GIAS specification defines, through the use of IDL, the interfaces, data types and error conditions that represent a geospatial information library. A GIAS based geospatial library has interfaces that allow a client to search and discover information (data sets/products) contained in the library, inquire about details of a particular data set/product and arrange for the delivery of the data set/product to another location or to another system. Also provided are interfaces to allow a client to nominate information to be included in the library. There are also interfaces to allow library-to-library interchange of information as well as interfaces that support management and control of the client-library interactions.

The GIAS specification does NOT define interfaces for functions such as: locating libraries with specific characteristics (this is the function of a Trading service), requests for the collection or acquisition of information not in a library (this is the function of a collection requirements system), management of the underlying communication and other infrastructure or requests for processing of information not directly related to the search or delivery of information (this is the function of the exploitation and production systems).

The definitions and semantics associated with the elements of the GIAS specification are intended to be as general and as broadly useful as possible. It is intended to be a description of any single implementation or system but is intended to allow great latitude in the design and implementation schemes for geospatial libraries. However, to ensure interoperability, all systems that must interoperate must make the same interpretations concerning this general specification. A *profile* of the GIAS specification for the intended community of use is a critical supplement to the GIAS specification itself. A profile is a formal documentation of the specific interpretations, limits, and conventions chosen by the community of use. The USIGS community will be producing profiles of the GIAS specification that document these factors.

The following sections detail the interfaces, data types and error conditions that compose the GIAS interface definition.

All elements of the GIAS definition are contained in the GIAS module:

```
module GIAS
{
.... all GIAS elements ...

};
```

2.2 Data Types

2.2.1 USIGS Common Objects

In order to support interoperability among the components of the USIGS architecture, the most common or most broadly useful interfaces, data types and error conditions have been defined and collected into the USIGS Common Object Specification. The intent is for all USIGS specifications to draw upon the UCOS definitions when appropriate rather then redefine a common element. In order to support interoperability, the GIAS specification uses the definitions in the UCOS whenever they are appropriate. The specific UCOS entities that GIAS uses are detailed below. The definitions given are descriptions of how GIAS uses these entities and are not intended to replace the definitions specified in the UCOS. Only cases where the UCOS data type is used as an element of a GIAS data type are detailed below. For cases where the UCOS element is used in a GIAS operation, its intended use is defined in the text accompanying each operation. All GIAS operations are defined in section 2.3

2.2.1.1 NameValueList

```
typedef UCO::NameValueList PropertyList;
```

The NameValueList structure is re-used to hold the name value pairs (Properties) that are used to augment or clarify many of the operations of the RequestManager.

```
struct RegionData
  {
    GeoRegion boundaries;
    UCO::NameValueList region_data_header;
    any tile_data;
  };
```

The NameValueList *region_data_header* in the RegionData structure is used to hold information that describes the particular data set contained in *tile_data* . The information in *region_data_header* is analogous to the header information in a file that describes the content of the file.

```
struct RequestDescription
  {
    string user_info;
    string request_type;
    string request_info;
    UCO::NameValueList request_details;
  };
```

The NameValueList *request_details* in RequestDescription is used to hold information that fully describe the Request which with it is associated.

2.2.1.2 Rectangle

```
typedef UCO::Rectangle GeoRegion;
```

The GIAS specification uses the GeoRegion data type to define geospatial subsections of products or data sets. Currently the only type of subsection allowed is rectangular. The GIAS specification thus defines GeoRegion based on the UCOS Rectangle.

2.2.1.3 Status and State

The GIAS specification uses the State enumeration defined in UCOS identify the state of Request objects. An important concept defined by UCOS State is that of a TERMINAL and NON_TERMINAL states. As defined by UCOS, “A process in a TERMINAL state will remain in that state until action is taken that causes it to change state. A NON-TERMINAL state will eventually change to a TERMINAL state without any further action being taken.” See section 2.2.4 of the UCOS specification for further details.

2.2.2 GIAS Specific Data types

The GIAS specification defines a number of data types that are specific to the GIAS. The definitions of the specific types are given in the following sections.

2.2.2.1 General Data types

The GIAS defines a number of simple data types.

2.2.2.1.1 LibraryList, RequestList, ManagerList

```
typedef sequence < Library > LibraryList;

typedef sequence < string > ManagerList;

typedef sequence < Request > RequestList

typedef sequence < UseMode > UseModeList;
```

The GIAS specification defines four convenience structures that are a sequence of other defined types. LibraryList contains a sequence of references

of type Library. ManagerList contains a sequence of strings, where each strings is a name of a Manager type. RequestList contains a sequence of references of type request.UseModeList contains a sequence of UseModes (see below for the definition of UseMode).

2.2.2.1.2 LibraryDescription

```
struct LibraryDescription
  {
      string library_name;
      string library_description;
  };
```

The LibraryDescription structure contains the name of a specific library instance in the string *library_name* . The string library_description contains a human readable description of the library and its holdings.

2.2.2.1.3 Query

```
typedef string Query;
```

The structure Query is a string that holds a query for submittal to a catalog. The Boolean Query Syntax (BQS) which defines the syntax of this string is defined in Chapter 3.

2.2.2.1.4 UseMode

```
typedef string UseMode;
```

UseMode is a string that describes a purpose or intended use of a data set or product. It is used by the AccessManager to support client requests and monitoring of the readiness of products for their use.

2.2.2.2 RegionData

```
struct RegionData
  {
    GeoRegion boundaries;
    UCO::NameValueList region_data_header;
    any tile_data;
  };
```

The RegionData structure is used to contain a “tile” of geospatial data. A tile is defined to be a geospatially defined subsection of a product or data set. Its intended use is to allow a client to access a data set or product as a series of tiles rather then as a monolithic data set.

The RegionData structure is intended to be mostly self describing. It contains three related elements: the GeoRegion *boundaries* defines the geospatial extent of the data contained in this GeoRegion, the NameValueList *region_data_header* contains any information to describe the data in the GeoRegion (metadata) and the type any *tile_data* contains the actual data of the tile. The types of data contained in the type any and the specific metadata that describe it are defined in the appropriate GIAS profile.

2.2.2.3 QueryResults

```
typedef long NodeID;
typedef long EdgeID;

struct Node
  {
    NodeID id;
    string attribute_name;
    any value;
  };

struct Edge
  {
    EdgeID id;
    NodeID start_node;
    NodeID end_node;
    string relat_name;
    string relat_type;
  };

typedef sequence < Node > NodeList;
typedef sequence < Edge > EdgeList;

struct DAG
  {
    NodeList nodes;
    EdgeList edges;
  };

typedef sequence < DAG > QueryResults;
```

The QueryResults structure is used to contain a collection of results from a catalog query. Each individual result in this collection contains metadata that describes a data set or product and a reference to that data set or product in the

form of a Product or GeoProduct reference (See section 2.3.2 and 2.3.3) The following sections define the form and usage of the QueryResults structure.

The QueryResult structure is composed of a sequence of directed acyclic graphs (DAG's). Each DAG contains two types of information: data elements ('nodes") and relationships among these elements ("edges"). The nodes are contained in the sequence *NodeList* and the edges are contained in the sequence *EdgeList*.

Each node contains an attribute-value pair. The name of the attribute is contained in the string *attribute_name* and the value is contained in the type any *value* . Along with this attribute-value pair is a NodeID (a type long) which uniquely identifies this node in this DAG.

The relationships among the nodes are defined by the edges. Each edge defines a relationship between two nodes. Each Edge structure contains the NodeId's of the two Nodes being related (*start_node* and *end_node*), a string *relat_type* containing the type of relationship that exists between these two Nodes and a string *relat_name* that gives a name to this instance of relationship. Also contained in the Edge structure is an EdgeId (a type long) that uniquely identifies this edge in this DAG.

The set of results from a catalog query is expressed in a QueryResults structure by applying the following rules:

1) Each result consists of an identifier of a data set or product and a set of metadata elements. The identifier will be in the form of a Product or GeoProduct reference whichever is appropriate for the data set. (See sections 2.3.2 and 2.3.3) The metadata elements will each consist of an attribute name and a type and value for that attribute.

2) Each result is placed in its own DAG.

3) Each metadata element is placed in its own node (a Metadata Node") by setting the *attribute_name* of the node to reflect the name of the metadata element and by setting the *value* of the node to reflect the type and value of that metadata element. To insert the identifier into a node (" an Identifier Node") , the *attribute_name* of the node should be set to "GIAS_PRODUCT" and its value should be set to a Product or GeoProduct reference as appropriate. Every DAG in a QueryResults will contain one Identifier Node.

4) The number, type and name for the relationships in a DAG are dependent on the data model (if any) that underlies the database the generated the result and are thus implementation dependent. However, there are two broad categories of results, distinguished by their use of relationships.

In the first category ("data set like") a result simplies mirrors the physical implementation of the data base from which it was extracted In a data

set like result, only simple relationships like “Belongs to List” are needed. All metadata elements are considered to describe the product or data set identified in the result. All interpretation of the attributes and their relationships if any are the responsibility of the receiving client. Most simple catalog results will be of this form. The result is basically an unordered list of attribute-value pairs.

In the second category (“data model like”) a result is actually a projection of the data model that the database describes. In this category potentially many relationships of many types could be defined in order to convey how this data set reflects the data model. The result is a graph containing attribute-value pairs as nodes with relationships defined among these nodes.

In both cases, the metadata elements could be identical, the cases differing only by the inclusion of the relationships.

The choice of data set or data model like results is made independently by the client or server. In order to allow maximum flexibility in the design and implementation of servers and clients while still ensuring interoperability, a consistent interpretation of the QueryResults structure must be made in four different scenarios:

Client expects	Server delivers	Notes
Data set	Data set	
Data set	Data model	Client ignores unknown relationships
Data model	Data set	Client reconstructs data model
Data model	Data model	

The client is responsible for determining which of the scenarios is in effect. It must be able to do so by examining the DAG it has received and determining whether the server has sent a data set or a data model like result.

The following sections define the use of a QueryResults in data set and data model like forms.

2.2.2.3.1 Data set form

A result expressed in data set form is simply an unordered list of attribute value pairs. The only relationships allowed are those required to express multiple values of attributes. The result is placed in a DAG by placing each metadata element of the result into its own Node (“a Metadata Node”). The type and value of that node are the same type and value of the element of the list. The placement of the Nodes into the NodeList is arbitrary.

The only allowed relationship in a data set like result is that required to express a metadata element that has multiple values (cardinality of 1: N). In this case, an extra Node is created (“a List Node”). It will be given the *attribute_name* of “GIAS_LIST” and its *value* will be set to a type unsigned

short with a value of the number of Nodes this list includes. Each instance of a value of the multi-valued metadata element is placed in its own Node with the appropriate *attribute_name* and *value*. ("a List Element Node") A relationship is created between the List Node and each of the List Element Nodes. Each relationship will have the start_node set to the List Node and the end_node to one of List Element Nodes. The relationship type *relat_type* of each relationship will be set to "CONTAINS". The relationship name *relat_name* will be an empty string.

A client can identify a data set like result by searching for List Nodes in the Nodes of the QueryResults. A data set like result will have 0 (zero) or more List Nodes and **only** relationships of type "CONTAINS" that include one of the List Nodes.

In the example below, a single record with three metadata elements (CountryCode, Classification and Target) is converted into a DAG. One of the metadata elements (Target) is multi-valued. The DAG is created by creating 7 Nodes (1 Identifier Node, 1 List Node and 5 Metadata Nodes) and 3 relationships (one for each instance of a Target value). The value contained in the List Node is 3 indicating the 3 instances of Targets.

ImageID	CountryCode	Classification	Target
1234	US	U	T1,T2,T3

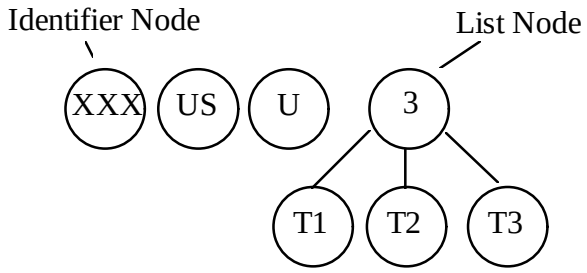

Fig 9. Simple DAG with Multi-valued Element

2.2.2.3.2 Data model form

A result in data model like form is intended to be a projection of the data model that underlies the database from which it was extracted. As such it will contain data types, relationship types and relationship names derived from that data model and thus is implementation dependent. However, in order to support interoperability, all clients should be able to perform the following with an arbitrary data model like QueryResults:

1) For each DAG, find all Nodes which are directly connected to the Identifier Node. These are the metadata elements that directly describe the data set or product.

2) Ignore without error any unknown data types, relationship types and relationship names.

2.2.2.4 RequestDescription

```
struct RequestDescription
  {
    string user_info;
    string request_type;
    string request_info;
    UCO::NameValueList request_details;
  };
```

The structure RequestDescription is used to describe the type and details of a request submitted for processing.

The RequestDescription structure is composed of 4 elements. The string user_info contains a message supplied by the submitting client, the contents of this message are completely determined by the client. The string *request_type* identifies the operation that was used to submit the request. The values and syntax of this element is defined in the appropriate GIAS profile. The string *request_info* contains any message the processing server wishes to return to the client concerning the specific request. It is intended to be human readable and is implementation dependent. The NameValueList *request_details* is intended describe the parameters of the operation that generated this request. The specific names and values in this element are dependent on the operation that initiated this request and will be defined in the appropriate GIAS profile.

2.3 Interfaces

2.3.1 Library

```
interface Library
  {

    ManagerList get_manager_types ();

    UCO::NameValueList access_criteria (in string manager_type)
    raises (UnknownManagerType);
```

```
    Manager get_manager (in string manager_type, in
    UCO::NameValueList access_criteria)
    raises (UnknownManagerType, BadAccessCriteria,
    BadAccessValue);

    LibraryDescription about ();
  };
```

The Library interface serves as the starting point for any interaction with the rest of the library. All capabilities of a library system are accessed through the manager objects it supports. The Library interface is the mechanism by which a client discovers and requests access to manager objects. The following operations are defined on the Library interface:

2.3.1.1 get_manager_types

```
ManagerList get_manager_types();
```

This operation allows a client to discover which managers are supported by a particular GIAS library. A ManagerList structure (section 2.2.2.1.1) is returned from a successful invocation of this operation. The ManagerList returned by this operation will contain the names of all manager types supported by this implementation. The manager names contained in this list are used with the access_criteria and get_manager operations defined below to specify the type of manger desired.

No user-defined exceptions are defined for this operation.

2.3.1.2 access_criteria

```
UCO::NameValueList access_criteria (in string manager_type)
raises (UnknownManagerType);
```

This operation allows a client to discover the attributes the client must provide to be allowed access to a given type of manager. The client invokes this method, supplying the type of manager desired in the parameter *manager_type.* A successful invocation returns a NameValueList containing the names of each attribute the client must correctly supply to be given access to a manager of the type specified. The value element of each NameValue in this list will contain one of two possible values expressed as a string: “MANDATORY” for an attribute that **must** be supplied for access to be granted or “OPTIONAL” for an attribute that could be supplied. This NameValueList is passed into the get_manager operation to actually gain access to the requested manager object (see below)

The exception UnknownManagerType is returned by this operation if the client has supplied a value of *manager_type* unknown or unsupported by this implementation.

2.3.1.3 get_manager

```
Manager get_manager (in string manager_type, in
UCO::NameValueList access_criteria)
raises (UnknownManagerType, BadAccessCriteria, BadAccessValue);
```

This operation is a request to be given access to a manager object. The client supplies the type of manager desired in *manager_type* and a NameValueList containing acceptable values for each mandatory attribute in *access_criteria.* (See the get_manager_types and access_criteria operations for details on determining acceptable values). A successful invocation will return a reference to an object of type Manager (see section 2.3.4). This reference should then be narrowed (cast) into a reference to an object of the specific manager type requested in *manager_type* . It can be assumed that all manager types supported by a GIAS implementation are derived (inherit) from type Manager. The client must know the correlation between the names given in the ManagerList and the object type to which that corresponds. Repeated calls to get_manager by the same client will result in new instances of Managers being returned. Also calls to get_manager by different client will always result in different Managers references being returned. That is the library system will not force clients to share Managers.

The exception UnknownManagerType is returned by this operation if the client has supplied a value of *manager_type* unknown or unsupported by this implementation. Supplying an unknown criteria in *access_criteria* will result in the BadAccessCriteria exception. Supplying an unacceptable value for an OPTIONAL attribute in access_criteria will result in the BadAccessValue exception. Supplying incorrect or unacceptable values for one or more MANDATORY attributes in *access_criteria* will result in the NO_PERMISSION system exception being returned. (See Appendix B for list of other system exceptions)

2.3.1.4 about

```
LibraryDescription about ();
```

This operation returns some descriptive information about the library. A successful invocation of this operation will return a populated LibraryDescription structure. (See section 2.2.2.1.2)

2.3.2 Product

```
interface Product
  {
  };
```

The Product interface serves a unique handle or identifier for a data set or product that resides in a library system. No methods are defined on this interface, its sole purpose is to serves as an identifier. All GIAS operations that require the identification of a specific data set or product, identify that data set or product by using the Product reference assigned to it.

Since there are no methods defined on this interface, no exceptions are defined.

2.3.3 GeoProduct

```
interface GeoProduct:Product
  {
  };
```

The GeoProduct interface serves the same purpose as the Product interface from which it is derived: a unique handle or identifier for a data set or product that resides in a library system. It extends this definition by requiring that data sets or products referred to with GeoProduct references have some type of geospatial coordinate system as a basis. GIAS operations that can successfully operate on data sets that do **not** have a geospatial coordinate systems as a basis will use Product references in their operation signatures. GIAS operations that **require** data sets with a geospatial coordinate system basis will use GeoProduct references in their operation signature. All operations that use Product in their signatures must also accept GeoProduct.

Since there are no methods defined on this interface, no exceptions are defined.

2.3.4 Manager

```
interface Manager
  {
    PropertyList get_properties (in UCO::NameList
desired_properties)
      raises (UnknownProperty);

    LibraryList get_libraries ();
             };
```

The Manager interface serves as the (abstract) root for all types of manager objects in the GIAS definition. It is abstract in the sense that a concrete Manager object by itself would serve no real purpose. Its real purpose is to define certain operations that are common to all types of manager objects in GIAS. Because these operations are common to all manager types, a client can use these common methods to interact with managers of unfamiliar type. The Manager interface defines the following operations:

2.3.4.1 get_properties

```
PropertyList get_properties (in UCO::NameList
desired_properties)
      raises (UnknownProperty);
```

This operation allows a client to discover the properties and the current values of those properties that describe a Manager. A client supplies the names of the properties of interest in the NameList *desired_properties* . A successful invocation of this operation returns a PropertyList (see section 2.2.1.1) which contains the current values of the requested properties. The PropertyList will contain one NameValue pair for each element supplied in the NameList *desired_properties* . The *name* in that NameValue pair will be the name as specified in *desired_properties* . The *value* associated with that *name* will be the current value of that property. The specific set of properties supported by a Manager is defined in the appropriate GIAS profile. However, the property “PropertyNames” is supported by all GIAS Managers. Invoking this method with the property “PropertyNames” as an element of the NameList *desired_properties* will, upon successful completion, return a PropertyList containing a single NameValue pair. This single entry will have the *name* “PropertyNames” and the *value* will be of type UCO::NameList where the elements of the NameList are all the properties supported or known to this Manager.

The exception UnknownProperty will be returned if the client has supplied one or more properties unknown or unsupported by this Manager. The *exception_details* element of the exception_info structure (See section 2.4.1) returned with the UnknownProperty exception will contain an explanation containing the names of all the unknown or unsupported properties supplied which caused the exception.

2.3.4.2 get_libraries

```
LibraryList get_libraries ();
```

This operation allows a client to determine which GIAS based library system(s) this Manager supports. A successful invocation of this operation will return a LibraryList structure (See section 2.2.2.1.1). This structure will contain an object reference of type Library for each library this Manager supports. There will always be at least one Library object reference in this list.

There are no user-defined exceptions defined on this operation.

2.3.5 RequestManager

```
interface RequestManager:Manager
  {

    RequestList list_active_requests ();

    void set_default_timeout (in long new_default)
      raises (ImplementationLimit);

    long get_default_timeout ();

    void set_timeout (in Request request, in long new_lifetime)
      raises (UnknownRequest, ImplementationLimit);

    void delete_request (in Request request)
      raises (UnknownRequest);
  };
```

The RequestManager interface serves to define operations common to all managers that use Request objects as part of their operations. This interface is abstract, like the Manager interface from which it is derived. Also like the Manager interface, these common operations allow a client to interact with unfamiliar forms of RequestManagers. The operations defined on RequestManager serve to allow clients to identify active requests and control their lifetimes.

Each Request being managed by a RequestManager has a limited lifetime. This lifetime is considered to begin when the processing it represents reaches a terminal state and ends when the timeout set for that particular request has elapsed. (See section 2.2.1.3 for a description of terminal states) After a Requests lifetime has expired a RequestManager is free to (but is not required to) delete that Request as well as all resources associated with that Request.

The RequestManager interface defines these operations:

2.3.5.1 list_active_requests

```
RequestList list_active_requests ();
```

This operation allows a client to determine what requests are being managed by this RequestManager. A successful invocation of this operation will return a RequestList structure (See section 2.2.2.1.1) . This structure will contain an object reference of type Request for each Request currently being managed by this RequestManager.

2.3.5.2 set_default_timeout

```
void set_default_timeout (in long new_default)
raises (ImplementationLimit);
```

This operation allows a client to set a default value (in seconds) of the lifetime of the Requests being Managed by this RequestManager. The client supplies the desired lifetime in *new_default.* Following successful invocation of this operation, all new Requests managed by this RequestManager will have a lifetime of *new_default* seconds. This operation has no effect on the lifetime of Requests that already exist at the time of invocation of this operation.

The exception ImplementationLimit will be returned if the client attempt to set a default lifetime that exceeds the maximum lifetime supported by this RequestManager implementation. The value of this maximum is implementation dependent and may vary over time.

2.3.5.3 get_default_timeout

```
long get_default_timeout ();
```

This operation allows a client to determine the current default lifetime for Requests initiated by this RequestManager. Successful invocation of this operation will return the current default lifetime of Requests in seconds.

2.3.5.4 set_timeout

```
void set_timeout (in Request request, in long new_lifetime)
raises (UnknownRequest, ImplementationLimit);
```

This operation allows a client to modify the currently set value for the lifetime of a Request. The client supplies the Request that is to have its lifetime modified in *request* and the desired value of its new lifetime in *new_lifetime.* Following successful invocation of this operation, the lifetime of Request *request* will be *new_lifetime* seconds. If the Request *request* has not reached a terminal state (See section 2.2.1.3), the lifetime will be *new_lifetime*

seconds beginning from the time it reaches a terminal state. If Request *request* is already in a terminal state when this operation is invoked (that is a portion of its lifetime has already elapsed), the lifetime of Request *request* will be *new_lifetime* seconds beginning from the time the set_timeout operation successfully completes.

The exception UnknownRequest will be returned if the client has supplied a Request unknown to this instance of RequestManager. The exception ImplementationLimit will be returned if the client attempt to set a default lifetime that exceeds the maximum lifetime supported by this RequestManager implementation. The value of this maximum is implementation dependent and may vary over time

2.3.5.5 delete_request

```
void delete_request (in Request request)
raises (UnknownRequest);
```

This operation allows a client to destroy a Request and free all resources associated with that Request. A client supplies the Request to be destroyed in *request*. Following successful invocation of this operation, the RequestManager is free to (but is not required to) immediately destroy Request *request* and to free all resources associated with that Request

The exception UnknownRequest will be returned if the client has supplied a Request unknown to this instance of RequestManager. After the RequestManager has destroyed the Request, attempts to invoke operations on that Request will return the OBJECT_NOT_EXIST system exception.

2.3.6 AccessManager

```
interface AccessManager
  {
    UseModeList list_use_modes ();

    boolean check_availability (in Product product, in UseMode
    use_mode)
    raises (UnknownProduct, UnknownUseMode, BadUseMode);

    MakeAvailableRequest make_available (in Product product, in
    UseMode use_mode, out long delay)
    raises (UnknownProduct, UnknownUseMode, Bad   UseMode);

  };
```

The AccessManager is an abstract interface that serves to define operations common to managers that allow clients to determine and control

the “availability” of a data set or product. “availability” is defined as the readiness of a data set or product to be used by the other operations on the manager. An AccessManager describes “availability” by defining one or more UseModes. A UseMode is a state or condition of a data set or product that indicates its readiness to be used by the AccessManager for a specific purpose. The following operations are defined on AccessManager:

2.3.6.1 list_use_mode

```
UseModeList list_use_modes ();
```

This operation allows a client to discover the UseModes supported by this AccessManager. A successful invocation of this operation returns a UseModeList containing all of the UseModes supported or known to this AccessManager.

There are no user-defined exceptions defined on the operation.

2.3.6.2 check_availability

```
boolean check_availability (in Product product, in UseMode
use_mode)
raises (UnknownProduct, UnknownUseMode, BadUseMode);
```

This operation allows a client to determine whether a data set or product is ready for a specific purpose. A client indicates the data set or product of interest and its desired use by supplying both a reference of type Product in *product* and its intended use as a UseMode in *use_mode.* A successful invocation of this operation will return a boolean that indicates whether or not the requested data set or product is currently available for the requested use. A boolean value of “TRUE” indicates the product is available. A boolean value of FALSE indicates that product is not currently available for the requested use. This operation does **not** effect the current availability of the requested data set or product.

The exception UnknownProduct will be returned if the client supplied a product reference unknown to this AccessManager. The exception UnknownUseMode will be returned if the client supplied a UseMode unknown or unsupported by this AccessManager. The exception BadUseMode is returned if the client supplied a UseMode that is inappropriate or unsupported for the particular data set or product supplied in *product*

2.3.6.3 make_available

```
MakeAvailableRequest make_available (in Product product, in
UseMode use_mode, out long delay)
raises (UnknownProduct, UnknownUseMode, BadUseMode);
```

This operation allows a client to submit a request to make a product available for a specific purpose. A client indicates the data set or product of interest and its desired use by supplying both a reference of type Product in *product* and its intended use as a UseMode in *use_mode.* A successful invocation of this operation returns two elements: in *delay* an estimate of the time (in seconds) it will take to place the specified product into the requested UseMode and a reference to a MakeAvailableRequest (see section 2.3.23).

The exception UnknownProduct will be returned if the client supplied a product reference unknown to this AccessManager. The exception UnknownUseMode will be returned if the client supplied a UseMode unknown or unsupported by this AccessManager. The exception BadUseMode is returned if the client supplied a UseMode that is inappropriate or unsupported for the particular data set or product supplied in *product*

The BadUseMode exception will also occur if the requested data set or product can never be made available in the requested UseMode.

2.3.7 GeoDataSetMgr

```
interface GeoDataSetMgr:RequestManager,AccessManager
  {

DisseminateRequest disseminate (in Product product, in
UCO::FileLocation location, in PropertyList properties)
raises (UnknownProduct, BadLocation, UnknownProperty,
BadPropertyValue);

DisseminateRequest get_subgeo (in GeoProduct product, in
GeoRegion region, in UCO::FileLocation location, in
PropertyList properties)
raises (UnknownProduct, BadGeoRegion, BadLocation,
UnknownProperty, BadPropertyValue);

  };
```

The GeoDataSetMgr is a concrete interface that provides operations that allow a client to request that a specific data set or product (or a geographically defined subsection of one) be delivered as a file to a specified location. The following operations are defined on this interface:

2.3.7.1 disseminate

```
DisseminateRequest disseminate (in Product product, in
UCO::FileLocation location, in PropertyList properties)
raises (UnknownProduct, BadLocation, UnknownProperty,
BadPropertyValue);
```

This operation allows a client to request the delivery of a complete data set or product as a file to a specified location. The client indicates the data set or product of interest by supplying the product reference for the desired data set in *product* . The client also indicates the location to which the data set is to be delivered by supplying a FileLocation structure in *location*. The client also describes any properties that further refine, effect or amplify this request by supplying their names and values in the ProperyList *properties.* The properties that are available or applicable to this operation are defined in the appropriate GIAS profile. A successful invocation of this operation will return a reference to a DisseminateRequest object. (see section 2.3.17).

The exception UnknownProduct will be returned if the client supplied a product reference unknown to this GeoDataSetMgr. The exception BadLocation will be returned if the client supplies a location description which is syntactically invalid, incomplete or specifies a location unknown or inaccessible by the GeoDataSetMgr. This does **not** require the GeoDataSetMgr to determine the validity of the user_name - password combination specified in *location* or the availability of space at *location* to return successfully. The exception UnknownProperty will be returned if the client has supplied one or more properties unknown or unsupported by this Manager. The *exception_details* element of the exception_info structure (See section 2.4.1) returned with the UnknownProperty exception will contain an explanation containing the names of all the unknown or unsupported properties supplied which caused the exception. The exception BadPropertyValue is returned if the client has supplied one or more values for properties which are inappropriate or exceed the allowed or expected values of that property. The *exception_details* element of the exception_info structure (See section 2.4.1) returned with the BadPropertyValue exception will contain an explanation containing the names and values of all the properties supplied which caused the exception as well as a description of the appropriate or acceptable values for those properties.

2.3.7.2 get_subgeo

```
DisseminateRequest get_subgeo (in GeoProduct product, in
GeoRegion region, in UCO::FileLocation location, in
PropertyList properties)
raises (UnknownProduct, BadGeoRegion, BadLocation,
UnknownProperty, BadPropertyValue);
```

This operation allows a client to request the delivery of a geographically defined subsection of data set or product as a file to a specified location. The client indicates the data set or product of interest by supplying the product reference for the desired data set in *product* . The client indicates the desired geographic subsection by supplying a GeoRegion in *region* . The client also indicates the location to which the data set is to be delivered by supplying a

FileLocation structure in *location*. The client also describes any properties that further refine, effect or amplify this request by supplying their names and values in the ProperyList *properties.* The properties that are available or applicable to this operation are defined in the appropriate GIAS profile. A successful invocation of this operation will return a reference to a DisseminateRequest object. (see section 2.3.17).

The exception UnknownProduct will be returned if the client supplied a product reference unknown to this GeoDataSetMgr. The exception BadGeoRegion will be returned if the client supplies a GeoRegion that is incomplete or describes a region that is not partially or completely contained in the specified product. The exception BadLocation will be returned if the client supplies a location description which is syntactically invalid, incomplete or specifies a location unknown or inaccessible by the GeoDataSetMgr. This does **not** require the GeoDataSetMgr to determine the validity of the user_name - password combination specified in *location* or the availability of space at *location* to return successfully. The exception UnknownProperty will be returned if the client has supplied one or more properties unknown or unsupported by this Manager. The *exception_details* element of the exception_info structure (See section 2.4.1) returned with the UnknownProperty exception will contain an explanation containing the names of all the unknown or unsupported properties supplied which caused the exception. The exception BadPropertyValue is returned if the client has supplied one or more values for properties which are inappropriate or exceed the allowed or expected values of that property. The *exception_details* element of the exception_info structure (See section 2.4.1) returned with the BadPropertyValue exception will contain an explanation containing the names and values of all the properties supplied which caused the exception as well as a description of the appropriate or acceptable values for those properties.

2.3.8 GeoFeatureMgr

```
interface GeoFeatureMgr:RequestManager,AccessManager
  {

FeatureRequest get_features (in GeoProduct product, in
UCO::FileLocation location, in PropertyList properties)
raises (UnknownProduct, BadLocation, UnknownProperty,
BadPropertyValue);

  };
```

The GeoFeatureMgr is a concrete interface that provides operations that allow a client to request that a thematically defined subsection of a specific

data set or product be delivered as a file to a specified location. The following operations are defined on this interface:

2.3.8.1 get_features

```
FeatureRequest get_features (in GeoProduct product, in
UCO::FileLocation location, in PropertyList properties)
raises (UnknownProduct, BadLocation, UnknownProperty,
BadPropertyValue);
```

This operation allows a client to request the delivery of a thematic subsection or slice of a complete data set or product as a file to a specified location. The client indicates the data set or product of interest by supplying the product reference for the desired data set in *product* . The client also indicates the location to which the data set is to be delivered by supplying a FileLocation structure in *location*. The client describes the particular set of features or thematic description desired by supplying their names and values in the ProperyList *properties* . In addition to those properties, the client also describes any other properties that further refine, effect or amplify this request by supplying their names and values in the ProperyList *properties.* The properties and the features or thematic descriptions that are available or applicable to this operation are defined in the appropriate GIAS profile. A successful invocation of this operation will return a reference to a FeatureRequest object. (see section 2.3.18).

The exception UnknownProduct will be returned if the client supplied a product reference unknown to this GeoDataSetMgr. The exception BadLocation will be returned if the client supplies a location description which is syntactically invalid, incomplete or specifies a location unknown or inaccessible by the GeoDataSetMgr. This does **not** require the GeoDataSetMgr to determine the validity of the user_name - password combination specified in *location* or the availability of space at *location* to return successfully. The exception UnknownProperty will be returned if the client has supplied one or more properties unknown or unsupported by this Manager. The *exception_details* element of the exception_info structure (See section 2.4.1) returned with the UnknownProperty exception will contain an explanation containing the names of all the unknown or unsupported properties supplied which caused the exception. The exception BadPropertyValue is returned if the client has supplied one or more values for properties which are inappropriate or exceed the allowed or expected values of that property. The *exception_details* element of the exception_info structure (See section 2.4.1) returned with the BadPropertyValue exception will contain an explanation containing the names and values of all the properties supplied which caused the exception as well as a description of the appropriate or acceptable values for those properties.

2.3.9 CreationMgr

```
  interface CreationMgr:RequestManager
  {

CreationRequest create (in UCO::FileLocation new_product, in
DAG creation_attributes, in PropertyList properties)
raises (BadLocation, UnknownCreationAttribute,
BadCreationAttributeValue, UnknownProperty, BadPropertyValue);

CreationRequest metadata_create (in DAG creation_attributes, in
PropertyList properties)
raises (UnknownCreationAttribute, BadCreationAttributeValue,
UnknownProperty, BadPropertyValue);

  };
```

The CreationMgr interface allows a client to nominate a data set or product to a library(s) for inclusion in those libraries holdings. This interface also allows a client to nominate the meta-data of a data set or product for inclusion without supplying the data set or product itself. This interface defines the following operations:

2.3.9.1 create

```
CreationRequest create (in UCO::FileLocation new_product, in
DAG creation_attributes, in PropertyList properties)
raises (BadLocation, UnknownCreationAttribute,
BadCreationAttributeValue, UnknownProperty, BadPropertyValue);
```

This operation allows a client to nominate a data set or product for inclusion in the holdings of a library(s). The data set or product nominated must be accompanied by the appropriate metadata. The client nominates a data set or product by supplying a FileLocation *new_product* that points to the data set or product being nominated. The metadata that must accompany this nomination may be supplied in one of two ways: 1) the file at the location *new_product* contains the data set **and all** the appropriate metadata 2) the file at location new_product contains the data set **and some** (to include none) of the metadata and *creation_attributes* contains the remainder of the appropriate metadata. (See section 2.2.2.3 for a details on the DAG datatype). If the first method of metadata submission is chosen a NULL value is supplied for *creation_attributes* . All metadata for the nominated product is then expected to be in file at location *new_product* If a non-NULL value is supplied for *creation_attributes* this indicates that the second metadata

submission method has been chosen and that the metadata for the nominated product is to be found in the file at location *new_product* **and** in the DAG *creation_attributes.* If the same metadata element appears in both the file and in the DAG, the value appearing in the DAG takes precedence and will be used for the nomination. Note that this requires servers to "edit" products submitted with conflicting metadata or risk serving out products with metadata that doesn't match that in the catalog. The definition of the metadata elements (their names and acceptable values or ranges, whether mandatory or optional and their mapping into and out of various file formats that may be nominated) to be described in the file or in the DAG are defined in the appropriate GIAS profile. The client also describes any properties that further refine, effect or amplify this request by supplying their names and values in the ProperyList *properties.* (The properties that are available or applicable to this operation are defined in the appropriate GIAS profile.) A successful invocation of this operation will return a reference to a CreationRequest object (see section 2.3.19).

The exception BadLocation will be returned if the client supplies a location description which is syntactically invalid, incomplete or specifies a location unknown or inaccessible by the GeoDataSetMgr. This does **not** require the GeoDataSetMgr to determine the validity of the user_name - password combination specified in *location* or the availability of space at *location* to return successfully. The exception UnknownCreationAttribute will be returned if the client has supplied a metadata element in the DAG *creation_attributes* that is unknown or unsupported by this CreationMgr. Note that a server will ignore unknown attributes in a file nominated. The *exception_details* element of the exception_info structure (See section 2.4.1) returned with the exception UnknownCreationAttribute will contain an explanation containing the names of all the unknown or unsupported elements supplied which caused the exception. The exception BadCreationAttributeValue will be returned if the client supplies a metadata element, whether in a file or in the DAG *creation_attributes* , with a inappropriate or invalid value. The *exception_details* element of the exception_info structure (See section 2.4.1) returned with the exception BadCreationAttributeValue will contain an explanation containing the names and values of all the metadata elements supplied which caused the exception as well as a description of the appropriate or acceptable values for those elements. The exception UnknownProperty will be returned if the client has supplied one or more properties unknown or unsupported by this CreationMgr. The *exception_details* element of the exception_info structure (See section 2.4.1) returned with the UnknownProperty exception will contain an explanation containing the names of all the unknown or unsupported properties supplied which caused the exception. The exception BadPropertyValue is returned if the client has supplied one or more values for properties which are inappropriate or exceed the allowed or expected values of that property. The *exception_details* element of the exception_info structure

(See section 2.4.1) returned with the BadPropertyValue exception will contain an explanation containing the names and values of all the properties supplied which caused the exception as well as a description of the appropriate or acceptable values for those properties.

2.3.9.2 metadata_create

```
CreationRequest metadata_create (in DAG creation_attributes, in
PropertyList properties)
raises (UnknownCreationAttribute, BadCreationAttributeValue,
UnknownProperty, BadPropertyValue);
```

This operation allows a client to nominate the metadata of a data set or product for inclusion in a library(s) without supplying the data set or product itself. The client nominates the metadata by supplying all metadata elements in the DAG *creation_attributes.* The client also describes any properties that further refine, effect or amplify this request by supplying their names and values in the ProperyList *properties.* (The properties that are available or applicable to this operation are defined in the appropriate GIAS profile.) A successful invocation of this operation will return a reference to a CreationRequest object (see section 2.3.19).

The exception UnknownCreationAttribute will be returned if the client has supplied a metadata element in the DAG *creation_attributes* that is unknown or unsupported by this CreationMgr. The *exception_details* element of the exception_info structure (See section 2.4.1) returned with the exception UnknownCreationAttribute will contain an explanation containing the names of all the unknown or unsupported elements supplied which caused the exception. The exception BadCreationAttributeValue will be returned if the client supplies a metadata element in the DAG *creation_attributes* with a inappropriate or invalid value. The *exception_details* element of the exception_info structure (See section 2.4.1) returned with the exception BadCreationAttributeValue will contain an explanation containing the names and values of all the metadata elements supplied which caused the exception as well as a description of the appropriate or acceptable values for those elements. The exception UnknownProperty will be returned if the client has supplied one or more properties unknown or unsupported by this CreationMgr. The *exception_details* element of the exception_info structure (See section 2.4.1) returned with the UnknownProperty exception will contain an explanation containing the names of all the unknown or unsupported properties supplied which caused the exception. The exception BadPropertyValue is returned if the client has supplied one or more values for properties which are inappropriate or exceed the allowed or expected values of that property. The *exception_details* element of the exception_info structure (See section 2.4.1) returned with the BadPropertyValue exception will contain

an explanation containing the names and values of all the properties supplied which caused the exception as well as a description of the appropriate or acceptable values for those properties.

2.3.10 CatalogAccessMgr

```
interface CatalogAccessMgr:RequestManager
  {

QueryRequest submit_query (in string product_type, in Query
query, in PropertyList properties)
raises (UnknownProductType, BadQuery,BadQueryAttribute,
BadQueryValue, UnknownProperty, BadPropertyValue);

HitCountRequest hit_count (in string product_type, in Query
query, in PropertyList properties)
raises (UnknownProductType, BadQuery, BadQueryAttribute,
BadQueryValue, UnknownProperty, BadPropertyValue);

};
```

The CatalogAccessMgr allows a client to submit queries to search the catalog of holdings of a GIAS library. Each CatalogAccessMgr manages one or more sub-catalogs. Each subcatalog is composed of all the holdings of the same product type. A CatalogAccessMgr will therefore have one sub-catalog for each product type in its library. The CatalogAccessMgr defines the following operations:

2.3.10.1 submit_query

```
QueryRequest submit_query (in string product_type, in Query
query, in PropertyList properties)
raises (UnknownProductType, BadQuery, BadQueryAttribute,
BadQueryValue, UnknownProperty, BadPropertyValue);
```

This operation allows a client to submit a query to search a catalog of products. The client indicates the product type of interest by supplying the desired value in *product_type.* (The specific product types available and acceptable to a CatalogAccessMgr are defined in the appropriate GIAS profile.) The query, which defines the selection criteria for the products of interest, is defined by the Query *query.* The syntax and details of the Query data type are explained in Chapter 3. The client also describes any properties that further refine, effect or amplify this request by supplying their names and values in the ProperyList *properties.* (The properties that are available or applicable to this operation are defined in the appropriate GIAS profile.) A

successful invocation of this operation will return a reference to a QueryRequest object. (see section 2.3.21).

The exception UnknownProductType will be returned if the client has supplied a data type unknown or unsupported by this CatalogAccessMgr. The exception BadQuery will be returned if the Query specified by *query* is syntactically invalid. The exception BadQueryAttribute will be returned if the query contains an attribute unknown to the CatalogAccessMgr. The *exception_details* element of the exception_info structure (See section 2.4.1) returned with the exception BadQueryAttribute will contain an explanation containing the names of all the unknown query attributes supplied which caused the exception. The exception BadQueryValue is returned if the client has supplied one or more values for query attributes which are inappropriate or exceed the allowed or expected values of that attribute. The *exception_details* element of the exception_info structure (See section 2.4.1) returned with the BadQueryValue exception will contain an explanation containing the names and values of all the attributes supplied which caused the exception as well as a description of the appropriate or acceptable values for those attributes. The exception UnknownProperty will be returned if the client has supplied one or more properties unknown or unsupported by this CreationMgr. The *exception_details* element of the exception_info structure (See section 2.4.1) returned with the UnknownProperty exception will contain an explanation containing the names of all the unknown or unsupported properties supplied which caused the exception. The exception BadPropertyValue is returned if the client has supplied one or more values for properties which are inappropriate or exceed the allowed or expected values of that property. The *exception_details* element of the exception_info structure (See section 2.4.1) returned with the BadPropertyValue exception will contain an explanation containing the names and values of all the properties supplied which caused the exception as well as a description of the appropriate or acceptable values for those properties.

2.3.10.2 hit_count

```
HitCountRequest hit_count (in string product_type, in Query
query, in PropertyList properties)
raises (UnknownProductType, BadQuery, BadQueryAttribute,
BadQueryValue, UnknownProperty, BadPropertyValue);
```

This operation allows a client to determine the number of results (“hits”) that would be returned from a particular query. The operation parameters, properties and exceptions for this operation are identical in form and meaning to those of the submit_query operation defined above. A successful invocation of this operation returns a reference to a HitCountRequest object.

2.3.11 ArrayAccessMgr

```
interface ArrayAccessMgr:RequestManager,AccessManager
  {

GetRegionRequest get_region (in GeoProduct product, in
GeoRegion region, in PropertyList properties)
raises (UnknownProduct, BadGeoRegion, UnknownProperty,
BadPropertyValue);

  };
```

The ArrayAccessMgr provides operations that allow a client to request a geospatial defined subsection of a data set or product to be delivered directly to the requesting client. The expected use of this interface is to allow a client to interact with geospatial data sets that are too large to deliver to the client or are more efficiently handled by offering geospatially indexed random access to the data set. The operation defined on this interface are :

2.3.11.1 get_region

```
GetRegionRequest get_region (in GeoProduct product, in
GeoRegion region, in PropertyList properties)
raises (UnknownProduct, BadGeoRegion, UnknownProperty,
BadPropertyValue);
```

This operation allows a client to submit a request to deliver a geospatially defined subsection of a data set or product directly to the requesting client. The client indicates the data set or product of interest in the GeoProduct *product* and the specific subsection of that product in the GeoRegion *region* . The client also describes any properties that further refine, effect or amplify this request by supplying their names and values in the ProperyList *properties.* (The properties that are available or applicable to this operation are defined in the appropriate GIAS profile.) A successful invocation of this operation will return a reference to a GetRegionRequest object. (see section 2.3.20).

The exception UnknownProduct will be returned if the client supplied a product reference unknown to this ArrayAccessMgr. The exception BadGeoRegion will be returned if the client supplies a GeoRegion that is incomplete or describes a region that is not partially or completely contained in the specified product. The exception UnknownProperty will be returned if the client has supplied one or more properties unknown or unsupported by this ArrayAccessMgr. The *exception_details* element of the exception_info structure (See section 2.4.1) returned with the UnknownProperty exception will contain an explanation containing the names of all the unknown or unsupported properties supplied which caused the exception. The exception BadPropertyValue is returned if the client has supplied one or more values for

properties which are inappropriate or exceed the allowed or expected values of that property. The *exception_details* element of the exception_info structure (See section 2.4.1) returned with the BadPropertyValue exception will contain an explanation containing the names and values of all the properties supplied which caused the exception as well as a description of the appropriate or acceptable values for those properties.

2.3.12 ProductAccessMgr

```
interface ProductAccessMgr:RequestManager
  {

ParametersRequest get_parameters (in Product product, in
PropertyList property_list)
raises (UnknownProduct, UnknownProperty, BadPropertyValue);

};
```

The ProductAccessMgr interface provides operations that allow a client to determine characteristics about a specific data set or product. This interface defines the following operation:

2.3.12.1 get_parameters

```
ParametersRequest get_parameters (in Product product, in
PropertyList property_list)
raises (UnknownProduct, UnknownProperty, BadPropertyValue);
```

This operation allows a client to submit a request to determine the characteristics of a specific data set or product. The client supplies a reference to the data set of interest in Product *product* The client also describes any properties that further refine, effect or amplify this request by supplying their names and values in the ProperyList *properties.* (The properties that are available or applicable to this operation are defined in the appropriate GIAS profile.) A successful invocation of this operation will return a reference to a ParametersRequest object. (see section 2.3.25).

The exception UnknownProduct will be returned if the client supplied a product reference unknown to this ProductAccessMgr. The exception UnknownProperty will be returned if the client has supplied one or more properties unknown or unsupported by this ProductAccessMgr. The *exception_details* element of the exception_info structure (See section 2.4.1) returned with the UnknownProperty exception will contain an explanation containing the names of all the unknown or unsupported properties supplied which caused the exception. The exception BadPropertyValue is returned if the client has supplied one or more values for properties which are

inappropriate or exceed the allowed or expected values of that property. The *exception_details* element of the exception_info structure (See section 2.4.1) returned with the BadPropertyValue exception will contain an explanation containing the names and values of all the properties supplied which caused the exception as well as a description of the appropriate or acceptable values for those properties.

2.3.13 IngestMgr

```
interface IngestMgr:RequestManager
  {

IngestRequest bulk_pull(in string product_type, in UCO::FileSet
data_files, in UCO::AbsTime since_time, in PropertyList
property_list)
raises(UnknownProductType, BadLocation, BadTime,
ImplementationLimit,  UnknownProperty, BadPropertyValue);

IngestRequest bulk_push(in UCO::FileSet data_files, in
PropertyList property_list )
raises(BadLocation, UnknownProperty, BadPropertyValue);

  };
```

The IngestMgr provides operations that allow a library to exchange large amounts of metadata with another library. The exchange takes place by exchanging (pushing or pulling) a set of files containing the metadata between the libraries. The format of the files exchanged and the mapping of those file formats into and out of the library's implementation are outside the scope of the GIAS. The details of this file format and its mappings will be detailed in the appropriate GIAS profile. The operations defined on this interface are:

2.3.13.1 bulk_pull

```
IngestRequest bulk_pull(in string product_type, in UCO::FileSet
data_files, in UCO::AbsTime since_time, in PropertyList
property_list)
raises(UnknownProductType, BadLocation, BadTime,
ImplementationLimit, UnknownProperty, BadPropertyValue);
```

This operation places a request to pull all metadata concerning a specified product type since a specified time out of a library. The client (the initiating library) indicates the product type of interest in *product_type* and a time. This indicates that the initiating library is requesting all metadata

concerning products of type *product_type* that have been entered into the library since time *since_time.* The initiating library also indicates the desired location of the metadata file set that results in FileSet *data_files.* The initiating library also describes any properties that further refine, effect or amplify this request by supplying their names and values in the ProperyList *properties.* (The properties that are available or applicable to this operation are defined in the appropriate GIAS profile.) A successful invocation of this operation will return a reference to a IngestRequest object. (see section 2.3.26).

The exception UnknownProductType will be returned if the initiating library has supplied a data type unknown or unsupported by this IngestMgr. The exception BadLocation will be returned if the client supplies a location description which is syntactically invalid, incomplete or specifies a location unknown or inaccessible by the IngestMgr. This does **not** require the IngestMgr to determine the validity of the user_name - password combination specified in *location* or the availability of space at *location* to return successfully. The exception BadTime is returned if the initiating library specified an invalid or inappropriate time. The exception ImplementationLimit will be returned if the submitted request exceeds the capabilities of the receiving library implementation. The exception UnknownProperty will be returned if the client has supplied one or more properties unknown or unsupported by this IngestMgr. The *exception_details* element of the exception_info structure (See section 2.4.1) returned with the UnknownProperty exception will contain an explanation containing the names of all the unknown or unsupported properties supplied which caused the exception. The exception BadPropertyValue is returned if the client has supplied one or more values for properties which are inappropriate or exceed the allowed or expected values of that property. The *exception_details* element of the exception_info structure (See section 2.4.1) returned with the BadPropertyValue exception will contain an explanation containing the names and values of all the properties supplied which caused the exception as well as a description of the appropriate or acceptable values for those properties.

2.3.13.2 bulk_push

```
IngestRequest bulk_push(in UCO::FileSet data_files, in
PropertyList property_list )
raises(BadLocation, UnknownProperty, BadPropertyValue);
```

This operation allows a library (the initiating library) to notify another library (the receiving library) that a block of metadata is available to be ingested. The initiating library indicates the location of the file set containing the metadata in FileSet *data_files.* The initiating library also describes any

properties that further refine, effect or amplify this request by supplying their names and values in the ProperyList *properties.* (The properties that are available or applicable to this operation are defined in the appropriate GIAS profile.) A successful invocation of this operation will return a reference to a IngestRequest object. (see section 2.3.26).

The exception BadLocation will be returned if the client supplies a location description which is syntactically invalid, incomplete or specifies a location unknown or inaccessible by the IngestMgr. This does **not** require the IngestMgr to determine the validity of the user_name - password combination specified in *location* or the availability of space at *location* to return successfully. The exception UnknownProperty will be returned if the client has supplied one or more properties unknown or unsupported by this IngestMgr. The *exception_details* element of the exception_info structure (See section 2.4.1) returned with the UnknownProperty exception will contain an explanation containing the names of all the unknown or unsupported properties supplied which caused the exception. The exception BadPropertyValue is returned if the client has supplied one or more values for properties which are inappropriate or exceed the allowed or expected values of that property. The *exception_details* element of the exception_info structure (See section 2.4.1) returned with the BadPropertyValue exception will contain an explanation containing the names and values of all the properties supplied which caused the exception as well as a description of the appropriate or acceptable values for those properties.

2.3.14 ProfileMgr

```
interface ProfileMgr:RequestManager
  {

    ProfileRequest submit_profile (in string product_type, in
Query query, in PropertyList properties)
      raises (UnknownProductType, BadQuery,BadQueryAttribute,
BadQueryValue, UnknownProperty, BadPropertyValue);

  };
```

The ProfileMgr interface provides operations that allow a client to submit standing queries. A standing query is a query on a catalog that continually monitors the catalog for new entries that match the specified query attributes. This interface defines the following operation:

2.3.14.1 submit_profile

```
ProfileRequest submit_profile (in string product_type, in Query
query, in PropertyList properties)
```

```
raises (UnknownProductType, BadQuery,BadQueryAttribute,
BadQueryValue, UnknownProperty, BadPropertyValue);
```

This operation submits a profile (standing query) for processing. The format of the operation parameters, exceptions and properties for this operation are identical in form and meaning to those of the submit_query operation defined on the CatalogAccessMgr (See section 2.3.10).

2.3.15 VideoAccessMgr

The VideoAccessMgr is intended to provide operations that allow a client to access a video data set as a temporal stream as well as a geospatial data set. The requirements and design of this interface and methods are TBR.

2.3.16 Request

```
interface Request
{

RequestDescription about ();

void user_info (in string message)
raises (ImplementationLimit);

UCO::Status get_status ();

void cancel ();

void register_callback (in Callback callback)
raises (UnknownCallBack);

void free_callback (in Callback callback)
raises (UnknownCallBack, UnregisteredCallBack);

void register_email_callback (in UCO::EmailAddress address, in
string user_message)
raises (BadEmailAddress, ImplementationLimit);

RequestManager which_request_manager ();

  };
```

The Request interface is an abstract interface that defines those operations that are common to request objects. Each operation of a RequestManager (See section 2.3.5) returns a reference to a specialized Request object. All specialized Request objects are derived (inherit) from Request. This interface defines the following operations:

2.3.16.1 about

```
RequestDescription about ();
```

This operation returns a RequestDescription structure (See section 2.2.2.4) that describes the Request.

2.3.16.2 user_info

```
void user_info (in string message)
raises (ImplementationLimit);
```

This operation allows a client to provide information that describes the Request. The client supplies this information, in the form of a string in *message* . A successful invocation of this operation associates the clients message with the Request. This client supplied information can be accessed in the *user_info* element of the RequestDescription structure returned by the about operation (see above).

The ImplementationLimit exception will be returned if the client supplies a message that exceeds the maximum length allowed by the implementation. This maximum length is implementation dependent.

2.3.16.3 get_status

```
UCO::Status get_status ();
```

This operation returns the current status of the Request. A successful invocation returns a Status structure (see the UCOS document for details).

2.3.16.4 cancel

```
void cancel ();
```

This operation is used to terminate further processing of a Request. After successful invocation of this operation, all current and future processing associated with this Request is terminated.

Before	After
COMPLETED	COMPLETED
IN_PROGRESS	CANCELED
ABORTED	ABORTED
CANCELED	CANCELED
PENDING	CANCELED
OTHER	CANCELED

2.3.16.5 register_callback

```
void register_callback (in Callback callback)
raises (UnknownCallBack);
```

This operation allows a client to register a callback object with a Request. The purpose of a callback object is to provide a method to allow the Request to notify the client that processing of a Request has reached a terminal state. (See section 2.2.1.3 for a description of terminal states) A client can register zero or more Callback objects with a Request. Registering the same callback object more then once with the same Request results in that Callback being registered only once. The client indicates the callback object to be registered by supplying a reference to a Callback object in *callback* . Following successful invocation of this operation the callback specified will be associated with this Request (registered). When this Request reaches a terminal state, the appropriate operation(s) on the specified callback object will be invoked. (see section 2.3.27 for details of the operations invoked on the callback object).

The exception UnknownCallBack will be returned if the client supplies a reference to a Callback object that is unknown or unreachable by the Request.

2.3.16.6 free_callback

```
void free_callback (in Callback callback)
raises (UnknownCallBack, UnregisteredCallBack);
```

This operation allows a client to remove a callback previously registered with a Request. The client supplies a reference to the Callback that is to be de-registered. Following successful invocation of this operation, the Callback specified will no longer be registered with this Request.

The exception UnknownCallBack will be returned if the client supplies a reference to a Callback object that is unknown or unreachable by the Request. The exception UnregisteredCallBack will be returned if the client attempt to free a callback which has not previously been registered with this Request.

2.3.16.7 register_email_callback

```
void register_email_callback (in UCO::EmailAddress address, in
string user_message)
raises (BadEmailAddress, ImplementationLimit);
```

This operation allows a client to submit a request to be notified by email when a Request reaches a terminal state (See section 2.2.1.3 for a description of terminal states).

Note: There is currently no operation to free or remove and email callback. Operations to support that function will be added in a later version of this specification.

2.3.16.8 which_request_manager

```
RequestManager which_request_manager ();
```

This operation allows a client to discover which RequestManager is managing the Request. A successful invocation of this operation returns a reference to the RequestManager that is managing this Request. This reference can be narrowed (cast) into a more concrete type.

2.3.17 DisseminateRequest

```
interface DisseminateRequest:Request
  {
    void complete ();
  };
```

The DisseminateRequest is returned by invocations of the disseminate operation of the GeoDataSetMgr. This interface defines the following operation:

2.3.17.1 complete

```
void complete ();
```

This operation allows a client to complete processing of the DisseminateRequest. This operation blocks until the requested dissemination reaches a terminal state (See section 2.2.1.3 for a description of a terminal state)

2.3.18 FeatureRequest

```
interface FeatureRequest:Request
  {
    void complete ();
  };
```

The FeatureRequest is returned by invocations of the get_features operation of the GeoFeatureMgr. This interface defines the following operation:

2.3.18.1 complete

```
void complete ();
```

This operation allows a client to complete processing of the FeatureRequest. This operation blocks until the requested operation reaches a terminal state (See section 2.2.1.3 for a description of a terminal state)

2.3.19 CreationRequest

```
interface CreationRequest:Request
  {
    Product complete ();
  };
```

The CreationRequest is returned by invocations of the create and metadata_create operation of the CreationMgr. This interface defines the following operation:

2.3.19.1 complete

```
Product complete ();
```

This operation allows a client to complete processing of the CreationRequest. This operation blocks until the requested operation reaches a terminal state (See section 2.2.1.3 for a description of a terminal state) A successful invocation of this operation returns a Product reference to the newly created product.

2.3.20 GetRegionRequest

```
interface GetRegionRequest:Request
  {
    RegionData complete ();
  };
```

The GetRegionRequest is returned by invocations of the get_region operation of the ArrayAccessMgr. This interface defines the following operation:

2.3.20.1 complete

```
RegionData complete ();
```

This operation allows a client to complete processing of the GetRegionRequest. This operation blocks until the requested operation reaches a terminal state (See section 2.2.1.3 for a description of a terminal state). A successful invocation of this operation returns a RegionData structure contain the data of the region requested.

2.3.21 QueryRequest

```
interface QueryRequest:Request
  {

    void numHits (in long hits);

    QueryResults complete ();
  };
```

The QueryRequest is returned by invocations of the submit_query operation of the CatalogAccessMgr. This interface defines the following operations:

2.3.21.1 numHits

```
void numHits (in long hits);
```

This operation allows a client to set the number of results (“hits”) that are returned by invocations of the operation complete (see below). This operation also sets the number of hits accumulated by this QueryRequest before a callback is triggered.

2.3.21.2 complete

```
QueryResults complete ();
```

This operation allows a client to complete processing of the QueryRequest. This operation blocks until the number of results set by numHits has been accumulated or all results have been processed. A successful invocation of this operation returns a QueryResults structure containing results from the query. Subsequent invocations of this operation can be used to retrieve any remaining results. The number of results returned in this structure is determined by the value set in a invocation of numHits (see above). A retrieval that returns a number of results less then the value previously set by numHits indicates that all results have been retrieved. If numHits has not been called prior to the invocation of complete, the number

of results returned in the QueryResults structure is determined by a default value which is implementation dependent.

2.3.22 ProfileRequest

```
interface ProfileRequest:Request
  {

    void numHits  (in long hits);

    QueryResults complete ();
  };
```

The ProfileRequest is returned by invocations of the submit_profile operation of the ProfileMgr. This interface defines the following operations:

2.3.22.1 numHits

```
void numHits (in long hits);
```

This operation allows a client to set the number of results (“hits”) that are returned by invocations of the operation complete (see below). This operation also sets the number of hits accumulated by this ProfileRequest before a callback is triggered.

2.3.22.2 complete

```
QueryResults complete ();
```

This operation allows a client to complete processing of the ProfileRequest. This operation blocks until the number of results set by numHits has been accumulated or all results have been processed. A successful invocation of this operation returns a QueryResults structure containing results from the profile. Subsequent invocations of this operation can be used to retrieve any remaining results. The number of results returned in this structure is determined by the value set in a invocation of numHits (see above). A retrieval that returns a number of results less then the value previously set by numHits indicates that all results have been retrieved. If numHits has not been called prior to the invocation of complete, the number of results returned in the QueryResults structure is determined by a default value which is implementation dependent.

2.3.23 MakeAvailableRequest

```
interface MakeAvailableRequest:Request
  {
    void complete ();
  };
```

The MakeAvailableRequest is returned by invocations of the make_available operation of the AccessMgr. This interface defines the following operation:

2.3.23.1 complete

```
void complete ();
```

This operation allows a client to complete processing of the MakeAvailableRequest. This operation blocks until the requested operation reaches a terminal state. (See section 2.2.1.3 for a description of a terminal state) Upon successful completion of this operation, the MakeAvailableRequest is in a terminal state. If that terminal state is COMPLETED, the product or data set requested is in the UseMode requested.

2.3.24 HitCountRequest

```
interface HitCountRequest:Request
  {
    long complete ();
  };
```

The HitCountRequest is returned by invocations of the hit_count operation of the CatalogAccessMgr. This interface defines the following operation:

2.3.24.1 complete

```
long complete ();
```

This operation allows a client to complete processing of the HitCountRequest. This operation blocks until the requested operation reaches a terminal state (See section 2.2.1.3 for a description of a terminal state) A successful invocation of this operation returns a value that indicates the total number of results (“hits”) that would be returned if the query was executed.

2.3.25 ParametersRequest

```
interface ParametersRequest:Request
  {
    PropertyList complete ();
  };
```

The ParametersRequest is returned by invocations of the get_parameters operation of the ProductAccessMgr. This interface defines the following operation:

2.3.25.1 complete

```
PropertyList complete ();
```

This operation allows a client to complete processing of the ParametersRequest. This operation blocks until the requested operation reaches a terminal state (See section 2.2.1.3 for a description of a terminal state) A successful invocation of this operation returns a PropertyList structure that contains the properties and current values of those properties of the product or data set requested.

2.3.26 IngestRequest

```
interface IngestRequest:Request
  {
    void complete ();
  };
```

The IngestRequest is returned by invocations of the bulk_pull and bulk_push operations of the IngestMgr. This interface defines the following operation:

2.3.26.1 complete

```
void complete ();
```

This operation allows a client to complete processing of the IngestRequest. This operation blocks until the requested operation reaches a terminal state (See section 2.2.1.3 for a description of a terminal state) A successful invocation of this operation indicates that the file containing the metadata to be exchanged is available. For the bulk_pull operation this indicates that the metadata file is at the delivered to the location specified and is ready to be ingested by the pulling library. For the bulk_push operation this indicates that the metadata file has been found by the receiving library at the location specified. This operation does NOT indicate that the metadata file has

been successfully ingested by the receiving library. It merely indicates successful transfer of and access to the metadata file.

2.3.27 Callback

```
interface Callback
  {

    void callback_notify (in RequestDescription description);

    void re leased ();
  };
```

2.3.27.1 callback_notify

```
    void callback_notify (in RequestDescription description);
```

This operation notifies the Callback that it has been triggered or activated. A Request that has reached a terminal state (See section 2.2.1.3 for a description of terminal states) will trigger or activate all callbacks registered with it. (See section 2.3.16.5 for registration of callbacks with Requests) A Request will activate a callback by invoking this method and supplies a description of the triggering Request in *description* . (see section 2.2.2.4 for RequestDescription)

2.3.27.2 released

```
void released ();
```

This operation is invoked by the Request to indicate that the Callback will no longer be used (will not be notified in the future). This allows a client to release any resources associated with this callback.

2.4 Exceptions

2.4.1 Exception Information

All user-defined exceptions return a consistent argument called `exception_info`. User-defined exceptions are error conditions which are explicitly defined in the GIAS IDL.
The `exception_info` structure is defined as:

```
struct exception_info
{
  string exception_details;
};
```

The exception_details element is used to amplify the exact cause or condition that generated the exception. This element is meant to be a human-readable explanation of the exception and its cause.

There is also a standardized set of general purpose exceptions defined by industry which address the most common reasons for failures, including communication and network errors (See Appendix B).

The following sections detail the exceptions defined in this specification:

2.4.2 BadAccessCriteria

This exception indicates the client has supplied incomplete, invalid or otherwise unacceptable access criteria. The *exception_details* element of exception_info will identify the unacceptable access criteria submitted.

2.4.3 BadAccessValue

This exception indicates that one or more values supplied for access criteria was missing, incorrect or otherwise unacceptable. The *exception_details* element of exception_info will identify which access criteria element(s) submitted were unacceptable and if appropriate the acceptable values or range of values.

2.4.4 BadCreationAttributeValue

This exception indicates the client supplied a value for one or more creation attributes with an inappropriate type or invalid value (i.e. exceeded the allowed or expected range) The *exception_details* element of the exception_info structure will contain an explanation containing the names and values of all the unacceptable creation attributes supplied as well as a description of the appropriate or acceptable values for those elements.

2.4.5 BadEmailAddress

This exception indicates that the client supplied an email address that was syntactically incorrect, uninterpretable or unreachable from the server. The *exception_details* element of the exception_info structure will contain the email address as received by the server.

2.4.6 BadGeoRegion

This exception indicates that a GeoRegion data structure supplied by the client is incomplete or describes a region that is inappropriate for the processing requested (i.e. region is not contained in the requested product)

2.4.7 BadLocation

This exception indicates the client supplied a FileLocation structure that is syntactically invalid, incomplete or specifies a location unknown or inaccessible by the server.

2.4.8 BadPropertyValue

This exception indicates the client supplied a value for one or more properties which are inappropriate or exceed the allowed or expected values of that property. The *exception_details* element of the exception_info structure will contain an explanation containing the names and values of all the unacceptable properties supplied as well as a description of the appropriate or acceptable values for those properties.

2.4.9 BadQuery

This exception indicates that a query submitted by the client has improper syntax. This would include missing or mismatched delimiters, use of undefined operators or use of an operator inappropriate for an attribute. See Chapter 3 for a description of the BNF that describes the syntax for queries.

2.4.10 BadQueryAttribute

This exception indicates the client supplied one or more attributes unknown or unsupported by the server. The *exception_details* element of the exception_info structure will contain the unacceptable attributes.

2.4.11 BadQueryValue

This exception indicates the client supplied one or more values for query attributes which are inappropriate or exceed the allowed or expected values for that attribute. The *exception_details* element of the exception_info structure will contain an explanation containing the names and values of all the unacceptable attributes and their supplied value as well as a description of the appropriate or acceptable values for those attributes

2.4.12 BadTime

This exception indicates the client supplied a time value that is incomplete or exceeds the allowed or expected range of times. The *exception_details* element of the exception_info structure will contain the unacceptable time value supplied as well as the allowed or expected range of times.

2.4.13 BadUseMode

This exception indicates the client requested a UseMode that is inappropriate or unsupported for the product or conditions requested.

2.4.14 ImplementationLimit

This exception indicates the client requested an operation with a parameter that exceeds an implementation specific limit for that parameter. The *exception_details* element of the exception_info structure will contain a the name of parameter exceeded as well as the expected or allowed range of values for that parameter.

2.4.15 UnknownCallBack

This exception indicates the client supplies a reference to a Callback object that is unknown or unreachable by the Request.

2.4.16 UnknownCreationAttribute

This exception indicates the client supplied a creation attribute that is unknown or unsupported by the server. The *exception_details* element of the exception_info structure will contain an explanation containing the names of all the unknown or unsupported elements.

2.4.17 UnknownProductType

This exception indicates the client supplied data type unknown or unsupported by the server. The *exception_details* element of the exception_info structure will contain an explanation containing the name of the unknown or unsupported product type.

2.4.18 UnknownManagerType

This exception indicates the client requested a manager type unknown or unsupported by this implementation. The *exception_details* element of the exception_info structure will contain an explanation containing the name of the unknown or unsupported manager type.

2.4.19 UnknownProduct

This exception indicates that the client requested a product reference unknown to the server.

2.4.20 UnknownProperty

This exception indicates the client supplied one or more properties unknown or unsupported by the server. The *exception_details* element of the exception_info structure will contain an explanation containing the names of all the unacceptable properties supplied.

2.4.21 UnknownRequest

This exception indicates the client supplied a reference to a Request that is unknown to the server.

2.4.22 UnknownUseMode

This exception indicates the client supplied a UseMode unknown or unsupported by the server. The *exception_details* element of the exception_info structure will contain an explanation containing the name of the unacceptable UseMode supplied.

2.4.23 UnregisteredCallBack

This exception indicates the client attempted an operation that requires a registered callback with a reference to a callback that has not been previously registered.

3. Boolean Query Syntax

3.1 Overview

The Boolean query syntax (BQS) is a key part of the specification of the GIAS. The intent of the BQS is to formally define the syntax for queries made on geospatial catalogs. It is necessary to define the BQS in the GIAS specification to "decouple" the interfaces using for querying from the implementation details of the catalog. For example, the BQS allows a client to interact with a geospatial catalog in a uniform way regardless of the database or database type underlying the catalog implementation, the native query language of the database and the physical schema or data model of the database. This approach has the dual benefit of simplifying the generation of queries by the client while not constraining the catalog developers in the design choices for the implementation. The catalog implementors must however provide the capability to translate the BQS into whatever query language and physical schema they have chosen.

3.2 BQS Design

The BQS is based upon the concept of a attribute-operator-value triplet called a factor. Each factor represents a condition of interest to the client. These factors can be assembled into a complete query by relating the factors with the Boolean operators "and" and "or".

The formal definition of the syntax of the BQS, described in Backus-Naur Form (BNF), is detailed below.

3.3 BNF definition

The Backus-Naur Form (BNF) for the Boolean query syntax is show below.

```
query ::= [ "not" ] term  { "or" term }

term ::= factor { "and" factor }

factor::=( simple_attribute_name comp_op constant_expression )
         | ( geo_attribute_name geo_op geo_element)
         | ( geo_attribute_name rel_geo_op number  dist_units
"of" geo_element)
         | ( text_attribute_name [ "not" ] "like" quoted_string
)
         | ( "(" query ")" )
```

```
attribute_name ::= a member of the set of queryable attribute
names (defined in the appropriate GIAS profile)

simple_attribute_name ::= member of subset of attribute_name
for which boolean operators (comp_op) are allowed

geo_attribute_name ::= member of subset of attribute_name for
which geospatial operators are allowed

text_attribute_name ::= member of subset of attribute_name for
which string operators are allowed (“free text search”)

comp_op ::= "=" | "<" | ">" | "<>" | "<=" | ">="

constant_expression ::= number | quoted_string

geo_op ::= “intersect” | “outside”|  “inside”

rel_geo_op ::= “within” | “beyond”

dist_units ::= “feet” | “meters” | “statute miles” | “nautical
miles” | “kilometers”

geo_element ::= point | polygon | rectangle | circle | ellipse
| line | polygon_set | 3dpoint

sign ::= “+” | “-” | “” // plus or minus or nothing

number ::=  sign n [ "."  [ n ] ]

n  ::= digit { digit }

digit ::=  “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8”
| ” 9”

quoted_string ::= "'" { character } "'"

character ::= “a”|”b”| ....

Del = “,” // Delimiter

latitude ::= number

longitude ::= number

altitude ::= number

hemi ::= “N” | “S” | “E” | “W”
```

```
DMS ::= [digit] digit digit  “:”  digit digit “:” digit digit
“.” digit  hemi

latlon ::= latitude Del longitude | DMS Del DMS

coordinate ::= latlon

point :: = “POINT” “(“ coordinate “)”

3dpoint ::= “3DPOINT” “(“  coordinate Del altitude “)”

polygon ::= “POLYGON” “(“ coordinate Del coordinate Del
coordinate {Del coordinate}“)”

rectangle ::=  “RECTANGLE” “(“ upper_left Del lower_right  “)”

upper_left ::= coordinate

lower_right ::= coordinate

circle ::=  “CIRCLE” “(“ coordinate Del radius “)”

radius ::= number

ellipse ::= “ELLIPSE” “(“ coordinate Del major_axis_len Del
minor_axis_len Del north_angle “)”

major_axis_len ::= number

minor_axis_len ::= number

north_angle ::= number

line ::= “LINE” “(“ coordinate Del coordinate { Del coordinate}
“)”

polygon_set ::= “POLYGON_SET” “(“ polygon { Del polygon} “)”
```

The BNF rules are augmented by the following constraint:
Wildcard expressions are allowed using the character "%" to denote a match with 0 or more characters. For example the query:

```
name like 'rob%'
```

would match the following strings:

```
'rob'  "robert'  'robin'
```

The "like" and "not like" operators are the only operators used for text expressions and the only operators supporting wildcards.

Wildcards can be used to implement the effect of many character matching operations, such as: contains, begins with, ends with, not contains, not begins with, not ends with, and so forth.

For example:

```
    attribute like '%contains_this%'
    attribute like 'begins_with_this%'
    attribute like '%ends_with_this'
    attribute not like '%will_not_contain_this%'
    attribute not like
'will_not_begin_with_this%'
    attribute not like '%will_not_end_with_this’
```

4. Appendix A: GIAS IDL

```
//**************************************************************
**
//*
//*           The Geospatial and Imagery Access Service
//*
//*
//*     Description: Defines the data types and interfaces
needed //*      to support search, retrieval and access to
geospatial data //*         such as images, maps charts and their
supporting data
//*
//*
//*
//*
//*     History:
//*     Date        Author      Comment
//*     -----       --------    ------------
//*     15 May 97   D. Lutz     Initial release for review
//*     2 July 97   D. Lutz     Released for TEM Review
//*     11 July 97  D. Lutz     Changes based on 2 July TEM
//*     18 July 97  D. Lutz     Released for NIMA CCB
//*
//*     Notes
//*     -------
//*     NONE
//*
//*
//**************************************************************
**


//**************************************************************
**
//*     The USIGS Common Object Specification (UCOS)  contains
//*     all the basic data types and interfaces common across
//*     USIGS
//**************************************************************
**

#include "ucos.idl"


//**************************************************************
**
//*
//*     Module GIAS
//*
```

```
//*
//*     Description: The main module for the Geospatial &
Imagery
//*     Access Service
//*
//*
//************************************************************
**

module GIAS
{

//Forward references for all interfaces, just for convenience

// The Library itself
  interface Library;

// The data-like objects
  interface Product;
  interface GeoProduct;

// Abstract classes that help define the managers
  interface Manager;
  interface RequestManager;
  interface AccessManager;

// Specific managers defined
  interface GeoDataSetMgr;
  interface GeoFeatureMgr;
  interface CreationMgr;
  interface CatalogAccessMgr;
  interface ProfileMgr;
  interface ArrayAccessMgr;
  interface ProductAccessMgr;
  interface IngestMgr;
//interface VideoAccessMgr;

// The abstract request objects
  interface Request;

// Specific requests defined
  interface DisseminateRequest;
  interface FeatureRequest;
  interface CreationRequest;
  interface GetRegionRequest;
  interface QueryRequest;
  interface ProfileRequest;
  interface MakeAvailableRequest;
  interface HitCountRequest;
  interface ParametersRequest;
```

```
  interface IngestRequest;

// A general callback object
  interface Callback;


//*************************************************************
**
//*      GIAS specific data types
//*************************************************************
**

  typedef sequence < Library > LibraryList;

struct LibraryDescription
    {
      string library_name;
      string library_description;
    };

typedef UCO::NameValueList PropertyList;

  typedef UCO::Rectangle GeoRegion;

  struct RegionData
    {
      GeoRegion boundaries;
      UCO::NameValueList region_data_header;
      any tile_data;
    };


  typedef string Query;

  typedef long NodeID;
  typedef long EdgeID;

  struct Node
    {
      NodeID id;
      string attribute_name;
      any value;
    };

  struct Edge
    {
      EdgeID id;
      NodeID start_node;
      NodeID end_node;  .
      string relat_name;
```

```
        string relat_type;
      };

    typedef sequence < Node > NodeList;
    typedef sequence < Edge > EdgeList;

    struct DAG
      {
        NodeList nodes;
        EdgeList edges;
     };

    typedef sequence < DAG > QueryResults;

  // end QueryResults

    typedef sequence < Request > RequestList;
    typedef sequence < string > ManagerList;

    typedef string UseMode;
    typedef sequence < UseMode > UseModeList;

    struct RequestDescription
      {
        string user_info;
        string request_type;
        string request_info;
        UCO::NameValueList request_details;
       };


  //************************************************************
  **
  //*                      The Exceptions
  //************************************************************
  **

  struct exception_info
  {
    string exception_details;
  };

    exception BadAccessCriteria {exception_info info;};
    exception BadAccessValue {exception_info info;};
    exception BadCreationAttributeValue {exception_info info;};
    exception BadEmailAddess {exception_info info;};
    exception BadGeoRegion {exception_info info;};
    exception BadLocation {exception_info info;};
    exception BadPropertyValue {exception_info info;};
    exception BadQuery {exception_info info;};
```

```
  exception BadQueryAttribute {exception_info info;};
  exception BadQueryValue {exception_info info;};
  exception BadTime {exception_info info;};
  exception BadUseMode {exception_info info;};
  exception ImplementationLimit {exception_info info;};
  exception UnknownCallBack {exception_info info;};
  exception UnknownCreationAttribute {exception_info info;};
  exception UnknownProductType {exception_info info;};
  exception UnknownManagerType {exception_info info;};
  exception UnknownProduct {exception_info info;};
  exception UnknownProperty {exception_info info;};
  exception UnknownRequest {exception_info info;};
  exception UnregisteredCallback {exception_info info;};
  exception UnknownUseMode {exception_info info;};


//************************************************************
**
//*                              The Interfaces
//************************************************************
**


//************************************************************
**
//*      interface GIAS::Library.
//*
//*     Description: This object represents a Library. It
//*     provides methods to discover and acquire manager
objects,
//*     which provide access to all the functionality of this
//*     Library
//*
//************************************************************
**

  interface Library
  {

    ManagerList get_manager_types ();

    UCO::NameValueList access_criteria (in string manager_type)
    raises (UnknownManagerType);

    Manager get_manager (in string manager_type, in
    UCO::NameValueList access_criteria)
    raises (UnknownManagerType, BadAccessCriteria,
    BadAccessValue);

    LibraryDescription about ();
```

```
  };



//*************************************************************
**
//*     Interface GIAS::Product.
//*      This object represents a dataset contained in a
Library.
//*      It serves as a unique identifier to a particular
//*      dataset.
//*
//*     No methods are defined on Product.
//*
//*************************************************************
**

  interface Product
  {
  };

//*************************************************************
**
//*     Interface GIAS::GeoProduct.
//*      This object represents a geospatial dataset contained
in
//*      an Library. It serves as a unique identifier to a
//*      particular dataset.
//*
//*     No methods are defined on GeoProduct.
//*
//*************************************************************
**

  interface GeoProduct:Product
  {
  };



//*************************************************************
**
//*     Interface GIAS::Manager
//*
//*    Description: This (abstract) object defines the basic
//*    functions common to all types of managers.
```

```
//*
//*
//*************************************************************
**

  interface Manager
  {
    PropertyList get_properties (in UCO::NameList
    desired_properties)
      raises (UnknownProperty);

    LibraryList get_libraries ();
  };


//*************************************************************
**
//*     Interface GIAS::RequestManager
//*     Derived from GIAS::Manager
//*
//*     Description: This (abstact) object defines the basic
//*     functions common to managers that uses methods that
//*     generate request objects.
//*
//*
//*************************************************************
**


  interface RequestManager:Manager
  {

    RequestList list_active_requests ();

    void set_default_timeout (in long new_default)
      raises (ImplementationLimit);

    long get_default_timeout ();

    void set_timeout (in Request request, in long new_lifetime)
      raises (UnknownRequest, ImplementationLimit);

    void delete_request (in Request request)
      raises (UnknownRequest);
  };


//*************************************************************
**
//*     interface GIAS:: AccessManager
```

```
//*
//*     Description: Provides functions to check and request
the
//*     availability of Library products for specific purposes
//*
//*****************************************************************
**

  interface AccessManager
  {

    UseModeList list_use_modes ();

    boolean check_availability (in Product product, in UseMode
    use_mode)
    raises (UnknownProduct, UnknownUseMode, BadUseMode);

    MakeAvailableRequest make_available (in Product product, in
    UseMode use_mode, out long delay)
    raises (UnknownProduct, UnknownUseMode, BadUseMode);

  };



//*****************************************************************
**
//*                               The Managers
//*
//*****************************************************************
**

//*****************************************************************
**
//*     interface GIAS:: GeoDataSetMgr
//*     Derived from GIAS::RequestManager and
GIAS::AccessManager
//*
//*     Description: Provides methods to request the
//*     dissemination of a whole Product (dataset) as a file.
//*
//*
//*
//*****************************************************************
**


  interface GeoDataSetMgr:RequestManager,AccessManager
  {
```

```
    DisseminateRequest disseminate (in Product product, in
    UCO::FileLocation location, in PropertyList properties)
    raises (UnknownProduct, BadLocation, UnknownProperty,
    BadPropertyValue);

    DisseminateRequest get_subgeo (in GeoProduct product, in
    GeoRegion region, in  UCO::FileLocation location, in
    PropertyList properties)
    raises (UnknownProduct, BadGeoRegion, BadLocation,
    UnknownProperty, BadPropertyValue);

  };


//************************************************************
**
//*      interface GIAS:: GeoFeatureMgr
//*     Derived from GIAS::RequestManager and
GIAS::AccessManager
//*
//*     Description: Provides methods to request the
//*     dissemination of a selected set of features from a
//*     GeoProduct (dataset) as a file.
//*
//*
//*
//*
//************************************************************
**


  interface GeoFeatureMgr:RequestManager,AccessManager
  {

    FeatureRequest get_features (in GeoProduct product, in
    UCO::FileLocation location, in PropertyList properties)
    raises (UnknownProduct, BadLocation, UnknownProperty,
    BadPropertyValue);

  };



//************************************************************
**
//*      interface GIAS:: CreationMgr
//*     Derived from GIAS::AccessManager
//*     Description: Provides methods to request/nominate the
```

```
//*     archiving and cataloging of a new product to a Library
//*
//*
//*************************************************************
**

  interface CreationMgr:RequestManager
  {

    CreationRequest create (in UCO::FileLocation new_product,
in  DAG creation_attributes, in PropertyList properties)
    raises (BadLocation, UnknownCreationAttribute,
    BadCreationAttributeValue, UnknownProperty,
    BadPropertyValue);

    CreationRequest metadata_create (in DAG creation_attributes,
    in Propert yList properties)
    raises (UnknownCreationAttribute, BadCreationAttributeValue,
    UnknownProperty, BadPropertyValue);

  };



//*************************************************************
**
//*      interface GIAS:: CatalogAccessMgr
//*     Derived from GIAS::AccessManager
//*
//*     Description: Provides methods to submit a query for
//*     processing
//*
//*
//*************************************************************
**


  interface CatalogAccessMgr:RequestManager
  {

    QueryRequest submit_query (in string product_type, in Query
    query, in PropertyList properties)
    raises (UnknownProductType, BadQuery, BadQueryAttribute,
    BadQueryValue, UnknownProperty, BadPropertyValue);

    HitCountRequest hit_count (in string product_type, in Query
    query, in PropertyList properties)
    raises (UnknownProductType, BadQuery,BadQueryAttribute,
    BadQueryValue, UnknownProperty, BadPropertyValue);
```

```
  };


//************************************************************
**
//*      interface GIAS::ProfileMgr
//*     Derived from GIAS::RequestManager
//*
//*     Description: Provides methods to submit a
//*     profile/standing query
//*
//*
//************************************************************
**


  interface ProfileMgr:RequestManager
  {

    ProfileRequest submit_profile (in string product_type, in
    Query query, in PropertyList properties)
    raises (UnknownProductType, BadQuery,BadQueryAttribute,
    BadQueryValue, UnknownProperty, BadPropertyValue);

  };




//************************************************************
**
//*      interface GIAS:: ArrayAccessMgr
//*     Derived from GIAS::RequestManager and
GIAS::AccessManager
//*
//*     Description: Provides methods to retrieve a "tile" out
of
//*     a GeoProduct into memory, where a tile is a
//*     geographically defined subregion
//*
//*
//************************************************************
**


  interface ArrayAccessMgr:RequestManager,AccessManager
  {
```

```
    GetRegionRequest get_region (in GeoProduct product, in
    GeoReg ion region, in PropertyList properties)
    raises (UnknownProduct, BadGeoRegion, UnknownProperty,
    BadPropertyValue);

  };



//*************************************************************
**
//*      interface GIAS:: ProductAccessMgr
//*     Derived from GIAS::AccessManager
//*
//*     Description: Provides methods to retrieve data about a
//*     specific data set
//*
//*
//*************************************************************
**



  interface ProductAccessMgr:RequestManager
  {

    ParametersRequest get_parameters (in Product product, in
    PropertyList property_list)
    raises (UnknownProduct, UnknownProperty, BadPropertyValue);


  };

//*************************************************************
**
//*      interface GIAS:: IngestMgr
//*     Derived from GIAS::RequestManager
//*
//*     Description: Provides methods to perform bulk transfers
//*     of data between libraries
//*
//*
//*************************************************************
**



  interface IngestMgr:RequestManager
  {
```

```
    IngestRequest bulk_pull(in string product_type, in
    UCO::FileSet data_files, in UCO::AbsTime since_time, in
    PropertyList property_list)
    raises(UnknownProductType, BadLocation, BadTime,
    ImplementationLimit,  UnknownProperty, BadPropertyValue);

    IngestRequest bulk_push(in UCO::FileSet data_files, in
    PropertyList property_list )
    raises(BadLocation, UnknownProperty, BadPropertyValue);

  };




//*************************************************************
**
//*      interface GIAS:: VideoAccessMgr
//*     Derived from GIAS::AccessManager
//*
//*     Description: Provides methods to retrieve video data
//*
//*     NOTE: This interface is TBR
//*************************************************************
**


//interface VideoAccessMgr : RequestManager, AccessManager {
  //};




//*************************************************************
**
//*      interface GIAS:: Request
//*
//*     Description: An (abstract) object that provides methods
//*     common to all forms of requests
//*
//*
//*************************************************************
**


  interface Request
  {
```

```
    RequestDescription about ();

    void user_info (in string message)
    raises (ImplementationLimit);

    UCO::Status get_status ();

    void cancel ();

    void register_callback (in Callback callback)
    raises (UnknownCallBack);

    void free_callback (in Callback callback)
    raises (UnknownCallBack, UnregisteredCallback);

    void register_email_callback (in UCO::EmailAddress address,
    in string user_ message)
    raises (BadEmailAddess, ImplementationLimit);

    RequestManager which_request_manager ();

  };



//*************************************************************
**
//*      interface GIAS:: DisseminateRequest
//*     Derived from GIAS::Request
//*     Description: Returned by calls to disseminate.
//*
//*
//*************************************************************
**

  interface DisseminateRequest:Request
  {
    void complete ();
  };

//*************************************************************
**
//*      interface GIAS:: FeatureRequest
//*     Derived from GIAS::Request
//*
//*     Description: Returned by calls to get_features
//*
//*
```

```
//*************************************************************
**


  interface FeatureRequest:Request
  {
    void complete ();
  };



//*************************************************************
**
//*      interface GIAS:: CreationRequest
//*     Derived from GIAS::Request
//*
//*     Description: Returned by calls to create
//*
//*
//*************************************************************
**


  interface CreationRequest:Request
  {
    Product complete ();
  };


//*************************************************************
**
//*      interface GIAS:: GetRegionRequest
//*     Derived from GIAS::Request
//*
//*     Description: Returned by calls to get_region
//*
//*
//*************************************************************
**


  interface GetRegionRequest:Request
  {
    RegionData complete ();
  };


//*************************************************************
**
```

```
//*      interface GIAS:: QueryRequest
//*     Derived from GIAS::Request
//*
//*     Description: Returned by calls to query
//*
//*
//*************************************************************
**



  interface QueryRequest:Request
  {
    void numHits (in long hits);
    QueryResults complete ();
  };


//*************************************************************
**
//*     interface GIAS:: ProfileRequest
//*     Derived from GIAS::Request
//*
//*     Description: Returned by calls to submit_profile
//*
//*
//*************************************************************
**


  interface ProfileRequest:Request
  {
    void numHits (in long hits)
    raises (ImplementationLimit);

    QueryResults complete ();
  };



//*************************************************************
**
//*      interface GIAS:: MakeAvailableRequest
//*     Derived from GIAS::Request
//*
//*     Description: Returned by calls to makeAvailable
//*
//*
//*************************************************************
**
```

```
  interface MakeAvailableRequest:Request
  {
    void complete ();
  };


//************************************************************
**
//*      interface GIAS:: HitCountRequest
//*     Derived from GIAS::Request
//*
//*     Description: Returned by calls to Hitcount
//*
//*
//************************************************************
**



  interface HitCountRequest:Request
  {
    long complete ();
  };


//************************************************************
**
//*      interface GIAS:: ParametersRequest
//*     Derived from GIAS::Request
//*
//*     Description: Returned by calls to get_parameters
//*
//*
//************************************************************
**

  interface ParametersRequest:Request
  {
    PropertyList complete ();
  };

//************************************************************
**
//*      interface GIAS:: IngestRequest
//*     Derived from GIAS::Request
//*
//*     Description: Returned by calls to bulk_push and
bulk_pull
```

```
//*
//*
//*****************************************************************
**

  interface IngestRequest:Request
  {
    void complete ();
  };









//*****************************************************************
**
//*      interface GIAS::Callback
//*
//*     Description: General callback interface
//*
//*     NOTE: The Callback interface is implemented on the
//*     "client" side to allow "servers" to notify clients of
//*     completion of requests.
//*
//*****************************************************************
**


  interface Callback
  {

    void callback_notify (in RequestDescription description);

    void released ();
  };

};              // end if module GIAS
```

5. Appendix B: Reference OMG Standard IDL

CORBA Standard Exceptions

```
#define ex_body {unsigned long minor; completion_status
completed;}

enum completion_status {COMPLETED_YES, COMPLETED_NO,
COMPLETED_MAYBE};
enum exception_type {NO_EXCEPTION, USER_EXCEPTION,
SYSTEM_EXCEPTION};

exception UNKNOWN ex_body;
exception BAD_PARAM ex_body;
exception NO_MEMORY ex_body;
exception IMP_LIMIT ex_body;
exception COMM_FAILURE ex_body;
exception INV_OBJREF ex_body;
exception NO_PERMISSION ex_body;
exception INTERNAL ex_body;
exception MARSHAL ex_body;
exception INITIALIZE ex_body;
exception NO_IMPLEMENT ex_body;
exception BAD_TYPECODE ex_body;
exception BAD_OPERATION ex_body;
exception NO_RESOURCES ex_body;
exception NO_RESPONSE ex_body;
exception PERSIST_STORE ex_body;
exception BAD_INV_ORDER ex_body;
exception TRANSIENT ex_body;
exception FREE_MEM ex_body;
exception INV_IDENT ex_body;
exception INV_FLAG ex_body;
exception INTF_REPOS ex_body;
exception BAD_CONTEXT ex_body;
exception OBJ_ADAPTER ex_body;
exception DATA_CONVERSION ex_body;
exception OBJECT_NOT_EXIST ex_body;
```

6. Appendix C - UML Diagrams

The GIAS IDL interface has been modeled using Unified Modeling Language (UML). A brief description of the notation used for the GIAS class diagrams was described in section 1.2. The purpose of this section is to provide a more detailed overview of UML to show the reader the "what" and "how" of the use of the various UML diagrams for analysis and modeling.

UML is based on three object-oriented modeling languages: 1) Object Modeling Technique (OMT) by James Rumbaugh; 2) Booch Method by Grady Booch; and 3) Object-oriented Software Engineering Method by Ivar Jacobson. Although all three methods had a large critical mass of users the authors were motivated to merge their modeling methods based on the following rationale:

1. Their methods were evolving toward each other and already shared many commonalties.
2. A common modeling language would greatly enhance communication between designers and implementors.
3. A common modeling language would greatly enhance portability amongst object-oriented analysis and design tool vendors.
4. A combination of the three methods would have a synergistic effect of combining lesson learned and addressing problems that the former method did not address well.

UML has been submitted as a standard modeling language to OMG and can be obtained as OMG documents ad/97-01-01 — ad/97-01-14. Based on the above rationale and potential for standardization was the justification used for using UML as the modeling language for GIAS.

UML distinguishes between the notions of model and diagram. A model contains all of the underlying elements of information about a system under consideration and does so independently of how those elements are visually presented. A diagram is a particular visualization of certain kinds elements from a model and generally exposes only a subset of those elements' detailed information. A given model element might exist on multiple diagrams, but there is but one definition of that element in the underlying model.

UML defines notation and semantics for the following diagrams:

- class diagrams - Is a collection of (static) declarative model elements, such as classes, types, and their relationships, connected as a graph to each other and to each other and to their contents. Class diagrams may be organized into packages either with their underlying models or as separate packages that build upon the underlying model packages.

- use-case diagrams - Is a graph of actors, a set of use cases enclosed by a system boundary communication (participation) associations between the actors and the use cased, and generalizations among the use cases.
- interaction diagrams
    - sequence diagrams - Shows objects participating in a set of interactions based on their “lifelines” and the messages that they exchange arranged in time sequence.
    - collaboration diagrams - Shows interactions amongst a set of objects.
- state diagrams - Is a bipartite graph of states and transitions. It shows the sequences of states that an object or an interaction goes through during its life in response to received stimuli, together with its responses and actions.
- component diagrams - Is a graph of components connected by dependency relationships. It shows aspects of implementation, including source code structure and run-time implementation structure.
- deployment diagrams - Is a graph of nodes connected by communication associations. It shows the configuration of run-time processing elements and the software components, processes, and objects that live on them.

7. Acronyms

API	Application Program Interface
BNF	Backus-Naur Form
BQS	Boolean Query Syntax
CAF	Catalog Access Facility
CIIF	Common Imagery Interoperability Facilities
CIIP	Common Imagery Interoperability Profile
CIIWG	Common Imagery Interoperability Working Group
CIO	Central Imagery Office
CORBA	Common Object Request Broker Architecture
COTS	Commercial off-the-shelf
DAG	Directed Acyclic Graph
GIAS	Geospatial & Imagery Access Services
GOTS	Government off-the-shelf
IAF	Image Access Facility
IAS	Image Access Services
IASS	Image Access Services Specification
IDF	Imagery Dissemination Facility
IDL	Interface Definition Language
ISO	International Standard Organization
NIMA	National Imagery and Mapping Agency
OMG	Object Management Group
PNF	Profile and Notification Facility
TBD	To Be Determined
TBR	To Be Resolved
UCOS	USIGS Common Object Specification
UIP	USIGS Interoperability Profile
UML	Unified Modeling Language
USIGS	United States Imagery and Geospatial System

8. Points of Contact

USIGS Architecture Integration Group

Ron Burns, National Imagery and Mapping Agency
Phone: (703) 808-0891
FAX: (703) 808-0531
Email: BurnsR@nima.mil

Joe Wesdock, National Imagery and Mapping Agency
Phone: (301) 227-3110 x428
Email: WesdockJ@nima.mil

Project Lead for MITRE Interface Definition Support

John Polger, National Imagery and Mapping Agency
Phone: (202) 863-3004
FAX: (202) 488-0271
Email: PolgerJ@nima.mil

NIMA Libraries Interface Definition

Charlie Green, Sierra Concepts, Inc.
Phone: (610) 347-0602
FAX: (610) 347-0602
Email: cpg.sci@mindspring.com

GIAS Specification & Support, and RFCs

Dave Lutz, The MITRE Corporation
Phone: (703) 883-7848
FAX: (703) 883-3315
Email: dlutz@mitre.org

USIGS Interoperability Profile (UIP)

Bill Nell, Lockheed Martin Management & Data Systems
Phone: (610) 531-6012
FAX: (703) 962-3698
Email: William.H.Nell@lmco.com